NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Human Resources Recommender system based on discrete variables

Dina Sarovska

Dissertation report presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

HUMAN RESOURCES RECOMMENDER SYSTEM BASED ON DISCRETE VARIABLES

by

Dina Sarovska

Dissertation report presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

Advisor: Roberto Henriques

July 2021

ACKNOWLEDGEMENTS

I would like to express immense gratitude to everyone around me for all the support during the course of this bittersweet journey. The past two years have been quite challenging in maintaining a balance between life, work, and a master thesis that was not in my domain of expertise, but was nevertheless worth it!

To my supervisor, Ph.D. Roberto Henriques, I would like to express gratitude for accepting my proposal with enthusiasm and motivating me to proceed forward. Thank you for the overall guidance during this work and look forward to our further collaboration.

To my former colleagues, Tiago Pereira and Tiago Barroca, enormous gratitude for elaborating the idea and providing me with the grounds of this work. Special thanks to Tiago Pereira for helping me setup StarSpace, a challenge I did not forsee.

To my friends (DDT), you know who you are, thanks for the moral support at any given moment. Cheers to celebrating all our successes in life together.

To my partner Luís Castro, whom I can not thank enough for all the support provided, both technically and mentally. Special gratitude for all the understanding and all the encouragement in the moments I was doubting myself. Great appreciation for the long and exhaustive discussions day and night. I thank you tremendously.

Most importantly, I would like to express a great amount of appreciation for my dear family, for always supporting me no matter what, despite being thousands of kilometers away. Thanks for never doubting I would manage to surpass whatever obstacle I had along the way. I owe my successes to you.

To all of you, thank you!

ABSTRACT

Natural Language Processing and Understanding has become one of the most exciting and challenging fields in the area of Artificial Intelligence and Machine Learning. With the rapidly changing business environment and surroundings, the importance of having the data transformed in such a way that makes it easy to interpret is the greatest competitive advantage a company can have. Having said this, the purpose of this thesis dissertation is to implement a recommender system for the Human Resources department in a company that will aid the decision-making process of filling a specific job position with the right candidate. The recommender system fill be fed with applicants, each being represented by their skills, and will produce a subset of most adequate candidates given a job position. This work uses StarSpace, a novelty neural embedding model, whose aim is to represent entities in a common vectorial space and further perform similarity measures amongst them.

KEYWORDS

Text Mining, Resume parser, Word Embeddings, Recommender system, StarSpace

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

NLP	Natural Language Processing
t-SNE	t-Distributed Stochastic Neighbor Embedding
BoW	Bag-of-words
SGD	Stochastic Gradient Descent
SG	Skip-gram
CBOW	Continuous-bag-of-words
RMSE	Root Mean Square Error
LHS	Left-hand side
RHS	Right-hand side

1. INTRODUCTION

It has been a known fact that the most significant competitive advantage a company can possess is the ability to produce insights on its own data, improving the quality of the overall decision making. When data is structured (i.e., it is stored in a fixed field format), performing analysis on it is made more accessible, facilitating any form of pattern recognition or classification for business-driven analytics. However, the data can be of various types (such as documents, social media streams, databases etc.), most of the time being semi-structured or even unstructured. Transforming the unstructured data into something valuable and easy to interpret is probably one of the biggest challenges companies face nowadays and is one of their most significant necessities.

The pillars of every company are its resources, and therefore, it is essential always to have the right people at the right time. Recruiters often struggle to go through a lengthy process of either finding people on LinkedIn or going through applicant resumes which are often big in volumes. This makes the recruitment process much more complex and time-consuming. Although a lot of research and developments have been done in AI and, more specifically, in Machine Learning in the past few decades, some issues have not been fully resolved. One of the biggest thorns in AI is the ability of a computer to interpret and understand the natural language. This is one of the reasons why this challenging topic will be the focus of this master thesis.

Therefore, this work aims to develop a proof-of-concept application based on a recommender system whose goal is to aid the decision-making process of recruitment teams when filling specific job positions within a company. The followed methodology relies on a novelty neural network embedding model, which will work with text-based inputs representing job applicants and will be used to find an optimal subset of individuals whose profile is the closest to the intended one. This is achieved by training this model to accurately produce skills embeddings in a common vectorial space, based on profiles of real applicants, and further, represent other real candidates and job advertisements (what can be perceived as an ideal candidate) based on those embeddings. By encoding applicants and jobs based on these produced embeddings, the model should be able to apply similarity metrics among entities and quantify their degree of proximity, determining which candidates are more suited for a given role.

To achieve this goal, two instances of this model will be produced for two independent datasets: the first resulting from the scrapping of thousands of LinkedIn profiles of professionals working in IT companies or IT-related positions, and the second being comprised of over a thousand resumes collected from job applicants of an IT company. Since this work deals with text-oriented data, it will also characterize the extensive data cleaning. This process involves the extraction and structuring of information necessary to form inputs accepted by the recommender system.

To the best of our knowledge, this thesis relies on using the general-purpose neural embedding model, an approach that has not been followed in the domain of job-candidate recommendations. So, one of the challenges will also be to bring value and contribute to improving the state of the art regarding this specific area of applications.

This thesis will initially include a general overview of some of the background concepts used (in section 3), as well as a comprehensive state-of-the-art, followed by a detailed characterization of the proposed methodology (in section 4). Finally conclusions are presented in section 6.

2. STATE OF THE ART

2.1. TEXT MINING

Using the three constraints of NLP – lexical, syntactic and semantic analysis is a commonly used approach among most researchers for extracting information from unstructured data in resumes. (Sanyal, Hazra, Ghosh, & Adhikary, 2017) and (Sadiq, Ayub, Narsayya, Ayyas, & Tahir, 2016) propose a solution for parsing the information from the resume using the three abovementioned constraints, which include several steps. First, the text is divided into various segments, i.e., sections of a resume using a data dictionary of possible headings found in a resume. Each of these segments of text is defined by specific name entity recognizers – chunkers. Second, syntactic analysis is performed in order to check for the grammatical correctness of the sentences, and lastly, the semantic meaning of the sentences is deducted to determine if the sentence makes sense or not. In both works, the results from the parser are presented in a JSON format file containing all the extracted relevant information, which are further used to map the candidates with job positions.

The work presented in (Reza & Zaman, 2017) proposes a different solution to the same problem by suggesting two different methodologies for detecting the segments of the resumes. The first one is to convert the text to an HTML format from which the font size can be extracted and used as a measure for detecting sections in a resume. The second methodology is to use a data dictionary and build a parse tree that will indicate possible structural information of a heading in a resume.

(Kulkarni, 2017) suggests a framework for mining relevant entities from a text resume by showing how the separation of parsing logic from entity specification can be achieved. The author proposes a linguistic-based approach by using RegEx expressions in order to extract information from the resume. The framework includes a configuration file that specifies entities along with the patterns for extraction. However, the solution proposed is only limited to one format of a resume and is prone to errors when encountered with different formats of resumes.

The presented works overlap to some extent in their approaches, more specifically in the segment of detecting the sections of a resume, where all propose a usage of a predefined data dictionary of possible headings. Additionally, another approach suggests the parsing to be performed by converting the text to HTML format and further detecting the headings based on the font size. Furthermore, one of the works suggests parsing the data by using RegEx expressions, while others perform syntactic analysis.

2.2. MACHINE LEARNING ALGORITHMS

2.2.1. Supervised learning

(Gopalakrishna & Varadharajan, 2019) proposes the implementation of a resume classifier application by using supervised learning algorithms. With the use of an ensemble learning-based voting classifier, profiles of candidates are classified into a domain based on their interest, work experience and expertise mentioned by the candidate in the profile. This model includes techniques of topic modelling to introduce a new domain to the list of domains upon failing to achieve the threshold value of confidence for the classification of the candidate profile. The Stack-Overflow REST APIs are called for the profiles which fail on the confidence threshold test set in the application. The topics returned by

the APIs are subjected to topic modelling to obtain a new domain, on which the voting classifier is retrained after a fixed interval to improve the accuracy of the model. The results showed that the ensemble learning-based voting classifier performs very well in comparison to the individual classifiers while predicting most of the instances of the test data since the confidence of the model while categorizing the resumes is influenced by the majority of the votes cast by the individual classifiers.

(Ko, Park, & Seo, 2002) suggests a study in which they measure the importance of sentences using text summarization techniques that attribute weights to specific text features, thus representing a document as a vector of features. Two methods measure the importance of each of the sentences. The first method gives higher weights to the sentences which are more similar to the title and the second method measures the importance of terms by TF, IDF and χ^2 statistic values. The higher importance to the sentence is assigned to the one which has more important terms. In the end, the importance of the sentence is calculated as a combination of both methods. The experiments were conducted over two separate and labelled datasets on English and Korean language, and four different classifiers are used: Naive Bayes, Rocchio, K-NN and SVM. The F1 test is used as a performance measure, and the results show that this approach makes a significant improvement over these classifiers.

2.2.2. Unsupervised learning

(Schmidt, 2019) suggests a study that uses unsupervised learning to classify text, more specifically customer's feedbacks. The way it is done is by predefining 11 categories of text along with a description for each category. The basic idea in this study is to create a label vector by using classic word embeddings (static and word-level, meaning each word gets one pre-computed embedding), which represent the 11 labels as vectors and calculate the cosine distance of each label (A) to the user's feedback (B). The label with the highest similarity or higher than a certain threshold is assigned to the user's rating. The positive feedback is that the model learns in a more human-like way by understanding the actual meaning of each category it shall predict, and the approach can be implemented with no available data at hand since the word embeddings are publicly available and pretrained. On the other hand, the drawback of this approach is that there is no testing data at all to evaluate the actual performance of the model before usage. Therefore, the confidence level is set quite high to avoid classifications of the user's feedback incorrectly.

(Ko & Seo, Automatic Text Categorization by Unsupervised Learning, 2000) proposes a method that divides the documents into sentences and categorizes each sentence using keyword lists of each category and sentence similarity measure. This approach automatically creates training sentence sets using keyword lists of each category, which are later used for training and thus classifying text documents. For feature selection, χ^2 statistic is used and Naive Bayes classifier as a statistical text classifier. Keywords are defined for each category by hand, which contain special features of each category sufficiently. The keywords are chosen based on their category names and their synonyms. The average number of keywords for each category is 3 (e.g., Category = Religion, Keywords = Christianity, Catholicism, Buddhism). The sentences which contain pre-defined keywords of each category in their content words are chosen as the initial representative sentences. The remaining sentences are called unclassified sentences and are assigned to their related category by measuring similarities of the unclassified sentences to the representative sentences. There exist error sentences in the representative sentences. They do not have special features of a category even though they

contain the keywords of the category. To remove them, the representative sentences are ranked by computing the weight of each sentence as follows:

- Word weights are computed using Term Frequency (TF) and Inverse Category Frequency (ICF)
- The size of the vocabulary is selected by ranking words according to their χ^2 statistic with respect to the category.

This method automatically created training sets using keyword lists of each category and used them for training after which it classified text documents.

(Verma, 2017) proposes a method to extract relevant words from resumes using Term Document Matrix. The relevant words are categorized based on their impact on the resume. The categories are role, language, database and web related skills, software packages, tools and frameworks, OS and experience skills. Each of the words is given a different weight 1,2,3 according to the job position. Then, all words summed from the resume give the rank of that resume. The importance of these words has been calculated according to the cluster. On top of this, a ranking methodology has been applied to find the most suitable candidate. This study uses the K-means algorithm to cluster the resumes and ReliefF to find the important features in each cluster. According to the tests done by manually selecting resumes for certain job positions, the model has retrieved relevant resumes that fit the job description.

2.2.3. Recommender systems

(Van Essen, 2018) proposes an interactive beer recommender system based on word embeddings from free-text user reviews. The dataset used for this work consists of user reviews and ratings from a public website. The document and word embeddings are trained using StarSpace (Wu, et al., 2018) and word2vec (Mikolov, Chen, Corrado, & Dean, 2013), which are later used as input for two different SVMs. The system builds a user profile by allowing each user to provide a list of beers they like and dislike, which are later used to train the SVM and recommend the users a list of n beers. Both models are compared to baselines of similarity of the feature vectors, as well as popularity. The popularity baseline recommends the n most frequently reviewed beers. The similarity baseline however, is done by computing the cosine similarity between an input and the rest of the items the user rated and thus ranking them from most to least similar. The prediction from each input is combined using late fusion by means of a voting system. The embeddings are evaluated using artificial actors, which are users that have already rated/reviewed beers on the website. By using precision, the performance of each of these actors is measured in the system. The results show that the classifiers using word2vec embeddings have a higher precision on already consumed beers; however, StarSpace outperforms with the popularity baseline when testing all beers.

(Gornishka, Rudinac, & Worring, 2019) present an interactive multimodal learning system that allows search and exploration of social multimedia users in large networks, where similar users are recommended according to the chosen user of interest. The users, words and concepts are represented as embeddings using StarSpace. The goal of the study is to prove that these embeddings can be helpful not only for categorizing users but for automatically generating user and community profiles. By categorizing the users, it assists in annotating large datasets for communities that are frequently changing. Two datasets are used for the purpose of this work, publicly available data from

a neo-Nazi forum called Stormfront that contains user posts, and the second dataset containing tweets regarding extremist ideologies downloaded directly from Twitter. The approach is to generate both unimodal user representations and multimodal neural embeddings. The unimodal representations are done by using TFIDF, and the users are represented based on all modalities (entities, text, visual concepts and hashtags). The multimodal representations are created with StarSpace, by using the multilabel text classification training mode.

Two different setups, from multiple possible, are showcased in the paper for creating user and content embeddings. For both setups, training examples are generated per post, and the corresponding user is assigned a positive label. The main difference is in the way the input documents are generated. Setup StarSpace CW-U trains a model with two separate examples per post, bag-of-concepts (C) and bag-of-words (W) associated with a given user (U). Setup StarSpace W-UC, on the other hand, has bag-of-words (W) as examples, but every post is labelled with the concepts (C) associated with the given user (U). This setup implicitly minimizes the distance between a user and the concepts they use by simultaneously minimizing the distance between both entities and the post itself. The approach is validated by using artificial actors that stimulate a user's behaviour in the system, each of them covering a different task. All the experiments are done using a linear SVM with SGD as the interactive classifier. The results show that StarSpace outperforms the TFIDF representations in situations when the context and semantics are more relevant to the given task. On the other hand, TFIDF shows better results in encoding specific terms with not much context in need.

3. THEORETICAL BACKGROUND

This section will cover a literature review of the algorithms and tools used in this master thesis. It will give a theoretical explanation of what the concepts are and how they work. The algorithms showcased here are accountable for two of the main parts of the proposed approach: (1) how to embed text into a multidimensional vector space and (2) visualize these embeddings for further validation. The algorithm chosen for embedding the text is StarSpace by Facebook AI Research (Wu, et al., 2018). The embeddings produced from this model will be further visualized by using a dimensionality reduction technique called t-Distributed Stochastic Neighbor Embedding (t-SNE). Before going in-depth and explaining how both algorithms work, some introductory concepts will be presented as they are the base they rely on. Additionally, StarSpace will be compared against some algorithms mentioned in the State of the Art, and thus these algorithms will be briefly presented in this section as well.

3.1. BAG-OF-WORDS

A Bag-of-words model is a method of extracting features from a text by describing the occurrence of the words within a document (Raschka & Mirjalili, 2021). This model is used in document classification, where the frequency of occurrence of each word is treated as a feature for training a classifier (McTear, Callejas, & Griol, 2016). It consists of two elements, a vocabulary of known words and a measure of their presence. Any information regarding the order of the words is disregarded, hence why it is called a "bag" of words. The general idea behind this algorithm is that documents are similar if their content is similar.

3.2. WORD EMBEDDINGS

Word embeddings are a more advanced approach to the bag of words model. These embeddings are representations of words encoded as a real-valued vector in a vector space, where similar words are closer to each other (Gulli & Pal, 2017). At training time, each word is represented as a point in an embedding space and is being adjusted based on the words that surround the target word. Unlike one-hot encoded vectors, which are sparse and require thousands to millions of dimensions, these word embeddings reduce the dimensionality and represent the word with only tens to hundreds of dimensions. This subsection will present three different methods of word embeddings: 1) word2vec, 2) FastText and 3) StarSpace.

3.2.1. Word2vec

Word2vec by (Mikolov, Chen, Corrado, & Dean, 2013) is a two-layer neural network trained to reconstruct contexts of words from a large corpus of text. The output result from the model is a vector space consisting of hundreds of dimensions, where each word from the corpus is assigned a corresponding vector in the space. These vectors represent numerical embeddings of the words. The words are positioned in such a way that words that share a common context are in close proximity to each other in the vector space. This similarity between the vectors is evaluated using cosine similarity. Aside from the positional proximity of similar words, some techniques for measuring the quality of the embeddings are utilized, which prove that words can have multiple degrees of similarity (Mikolov, Yih, & Zweig, Linguistic Regularities in Continuous Space Word Representations, 2013). By using simple algebraic operations on the word embeddings, results show that $vector('King') - vector('Man') + vector('Woman') = vector('Queen')$.

Word2vec is implemented with two different architectures, and both can be utilized depending on the context of the problem aimed to be solved. The *Continuous-Bag-of-Words (CBOW)* architecture aims to guess the target word given a set of neighbouring context words, whereas the *Skip-gram (SG)* architecture attempts to guess the context neighbouring words given the target word. The architectures are displayed visually in Figure 3.1.

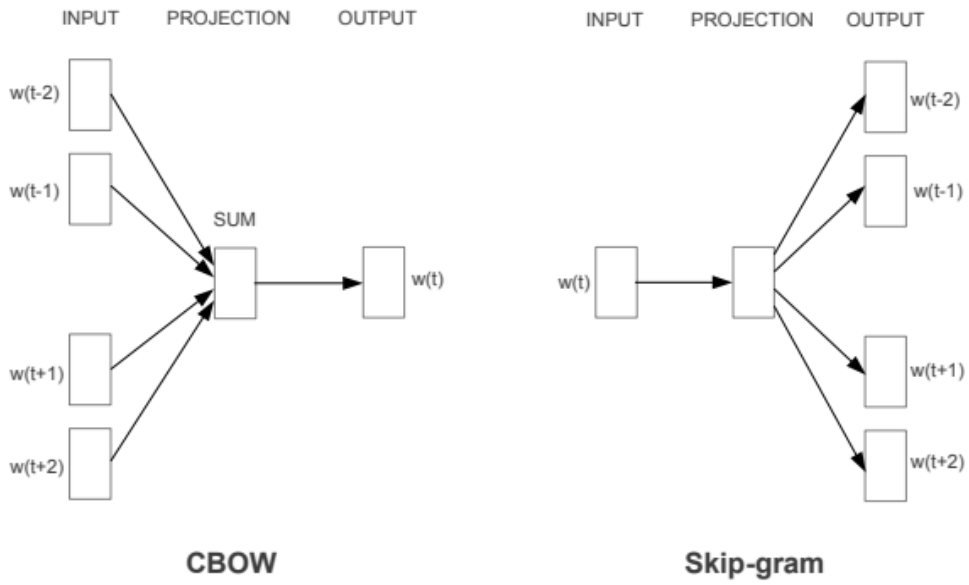


Figure 3.1 - Word2vec architectures, CBOW and Skip-gram

According to (Mikolov, Chen, Corrado, & Dean, 2013), the CBOW architecture is much faster to train than Skip-gram and has better accuracy for predicting frequent words. However, Skip-gram works well with small amounts of data and is able to represent well rare words or even phrases.

The objective function of the Skip-gram model, given a sequence of training words $w_1, w_2, w_3, \dots, w_T$, is to maximize the probability of any context word given the current target word.

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p\left(w_{t+j}|w_t\right) \qquad \text{(eq 3.1)}$$

where c is the number of context words used in training, T is the number of words in the vocabulary, w_t is the target word for which the model has to predict its neighbouring w_{t+1} context words and θ is the resulting vector representation of the words. The larger c is, the more training examples are covered, which leads to a higher accuracy at the cost of training time. Both model architectures use the hierarchical softmax as the activation function in the output layer of the neural network in order to reduce its computational complexity. The vocabulary is represented as a Huffman binary tree where each leaf of the tree is a single word, and each internal node represents the relative probabilities of the children nodes. Each word has a unique path from the root to its leaf. In this tree hierarchy, the frequent words are assigned short binary codes, which reduces the number of output units that need evaluation. However, at prediction time, the probabilities for all words need to be computed, which thus leads to a vast neural network. Since both models are trained using stochastic gradient descent and backpropagation, specific issues arise concerning the computational complexity of the neural network.

These issues were addressed by (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013), where a two-step approach is presented with an aim to reduce the complexity and additionally improve the quality of the resulting embeddings. Negative sampling is an approach based on Noise Contrastive Estimation (NCE) by (Gutmann & Hyvarinen, 2012) which persists that a model should differentiate data from noise by means of logistic regression. The general idea behind this method is similar to SGD. Namely, instead of updating all the weights of all thousands of observations every time, it only uses k of them which leads to an improved computational efficiency. In the context of the Skip-gram model of word2vec, the k negative samples generated are words which are not context words, meaning that these words are not the correct predictions for the given target word. The second approach suggested is subsampling of frequent words, which thus leads to a decrease in the number of training examples. Frequent words, more commonly referred to as stop words, can occur thousands of times in a large corpus of text but lack to provide any valuable information to the model. For each word of the vocabulary, a probability is computed that defines whether the word will be disregarded from the training dataset. The results have shown a much faster and significant improvement of the accuracy of learned embeddings of rare words.

3.2.2. FastText

One limitation with word2vec is that it does not consider the morphology of words and simply assigns a distinct vector to each word. This is a limitation for morphologically rich languages. FastText is a word embedding model proposed by (Bojanowski, Grave, Joulin, & Mikolov, 2017) that is an extension of word2vec and tends to solve this constraint. The approach is based on the continuous skip-gram architecture where each of the words is represented as a bag of character n-grams, hence providing sub word information (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). (Bojanowski, Grave, Joulin, & Mikolov, 2017) define a scoring function s which maps pairs of (word, context) to scores in $\mathbb{R}$. Given a dictionary of n-grams with size G, a word w is denoted by $G_w \subset \{1, \dots, G\}$, as the set of n-grams that appear in w. A vector representation z_g is associated to each n-gram g. This means that a word is represented as the sum of the vector representations of its n-grams. The following scoring function is obtained:

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c \qquad \text{(eq 3.2)}$$

The n-grams help capture the meaning of shorter words and thus enable the model to learn prefixes and suffixes. Once a word is represented with its n-grams, a skip-gram model is trained to learn the embeddings. The skip-gram model is considered a BoW model with a sliding window over a target word where the order of the n-grams is disregarded. Another advantage of FastText over Word2vec is that it works much better with rare words, meaning that even if a word was not present in the training dataset, it can be broken down into n-grams in order to get the embedding.

3.2.3. StarSpace

StarSpace is a general-purpose neural embedding model for learning entity embeddings in order to solve a wide range of use cases such as text and image classification, ranking entities, embedding graphs, learning word, sentence and document embeddings, as well as collaborative filtering-based and content-based recommendations (Wu, et al., 2018). The method works by learning entity embeddings from relationships amongst sets of entities. The general idea behind the model is to learn

how to represent entities of different types into a common vector space and further compare them against each other.

The model consists of learning entities that are described by a collection of discrete features (bag-of-features) generated from a fixed-length dictionary. A given entity, for instance, a user, can be described by the bag of documents, movies, items they have liked. As mentioned before, StarSpace allows for entities of different types to be compared against each other, in this case, a user to be compared with a document, movie, item etc.

In order to build embeddings in StarSpace, an entity needs to be represented in regards to its features. The dictionary of D features, denoted as F, is a $D \times d$ matrix where F_i indexes the i^{th} feature (row), obtaining the d-dimensional embedding to embed an entity a with $\sum_{i \in a} F_i$. In other words, each discrete feature from the dictionary of features is assigned a d-dimensional vector, and each entity composed of features is represented as a bag of features of the features in the dictionary, and their embeddings are learned implicitly. The goal of the model is to learn how to compare the entities amongst each other and thus minimize the following loss function:

$$\sum_{\substack{(a,b) \in E^+ \\ b^- \in E^-}} L^{batch}\big(sim(a,b), sim(a,b_1^-), \dots, sim(a,b_k^-)\big) \qquad \text{(eq 3.3)}$$

This function is composed of several components.

1. Generating positive entity pairs (a, b) from the set E^+.
2. Generating negative entities b_i^- from the set E^-. The model incorporates a k-negative sampling strategy (Mikolov, Chen, Corrado, & Dean, 2013) to choose k-negative pairs for each batch update. These samples are chosen randomly from the set of entities that can appear as b in the similarity function.
3. Similarity function $sim(\cdot,\cdot)$, which is implemented as a hyperparameter and can be either cosine similarity or inner dot product.
4. The loss function L^{batch} that compares the positive pair (a, b) with all the negative pairs $(a, b_i^-), i = 1, \dots, k$. It is also implemented as a hyperparameter with two available options: margin ranking loss and negative loss of softmax. However, the first has outperformed the latter in all the use cases showcased.

The model is optimized by using Stochastic Gradient Descent (SGD), meaning that each step is one sample from the positive entity pair set – E^+in the outer sum, using Adagrad (Duchi, Hazan, & Singer, 2011) and Hogwild (Niu, Recht, Re, & Wright, 2011) over multiple CPUs. Additionally, a max norm of the embeddings is enforced in order to restrict the embedding vectors learned to lie in space R^d as in other related works (Weston, Bengio, & Usunier, 2011).

Due to the wide possibility of the model assessing various tasks, the generators E^+ and E^- work differently according to the specific training mode chosen. The process of generating positive entity pairs (a, b) and negative entities b_i^- in the context of the problem proposed, i.e., content-based recommendation will be further explained in detail in the following section.

In the content-based recommendation use case, which is one of the many use cases showcased in the paper of StarSpace, five different methods are used as comparison baselines, both supervised and

unsupervised. Unsupervised methods include Word2vec, fastText, Tagspace and TFIDF, while an SVM ranker using either fastText embeddings or BoW features is used as a supervised method. The tests have shown superior results in favour of StarSpace regarding all other methods.

One of the main reasons for choosing StarSpace is the possibility of it handling featured labels. Featured labels mean that features can be used in order to represent labels. For instance, in StarSpace, a label can be a sentence that is represented by the words it contains. However, in the case of fastText, for example, a label needs to be a direct embedding like a word or a tag. This advantage of the model was key since the dataset used for assessing the problem proposed does not contain labels. Another useful feature that StarSpace provides is the option to assign weights to the words, which in turn helps the model to better learn the relationship dependencies amongst them.

3.3. TEXT NORMALIZATION METHOD

One of the most important methods applied in the data preprocessing phase is a text normalization method developed by (WB Advanced Analytics, 2017). The algorithm is based on the commonly used text similarity measure – cosine similarity. Cosine similarity measures the angle between the two n – dimensional vectors projected in a multi-dimensional space (Deep, 2020). However, (WB Advanced Analytics, 2017) detected a significant disadvantage with this similarity measure. The sklearn version calculates the similarity matrix and stores all the results instead of only considering the top N most similar, which would translate to a computationally expensive process. Therefore, (WB Advanced Analytics, 2017) developed their library, which outperforms SciPy and NumPy functions by 40% in implementing a sparse matrix multiplication and selecting the top N results above a given threshold. The method receives four hyperparameters. Two compressed sparse row (CSR) matrices, one matrix being the ground truth matrix and another matrix with the data aimed to be normalized. Additionally, a number n is provided, to retrieve only the top n results, as well as a threshold for similarity. The process starts by first storing the input data as n – grams. After the n – grams are produced, a Tfidfvectorizer is used in order to count their occurrences. By using the *sparse_dot_topn* function the top n most similar inputs above the given threshold are retrieved as result.

3.4. T-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) by (Van Der Maaten & Hinton, 2008) is a technique for visualizing high dimensional data in a lower-dimensional space such as 2D or 3D. The method is commonly used as it can detect non-linear relationships in the data. The first stage of the algorithm is to compute the Euclidian distances between each of the data points. Later on, these distances are transformed into conditional probabilities that represent the similarity between two data points. As (Van Der Maaten & Hinton, 2008) describe in the paper, the similarity of data point x_j to data point x_i is the conditional probability $p_{j|i}$ that x_i would pick x_j as its neighbor.

$$p_{j|i} = \frac{\exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|X_i - X_k\|^2}{2\sigma_i^2}\right)}$$

(eq 3.5)

The conditional probability of x_j to be in close proximity to x_i is represented by a Gaussian centered as x_i with a standard deviation of σ_i. From the conditional probabilities a joint probability distribution is calculated.

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \quad \text{(eq 3.4)}$$

The second stage of the algorithm is to perform a dimensionality reduction to two or three dimensions as well as calculate a joint probability distribution for all the data points. In order to create the join probability distribution, a t-distribution is used instead of the Gaussian distribution, and the reason for this is the heavy tails property of the t-distribution. This enables the distances between points in the high-dimensional space to be extreme in low-dimensional space and thus help prevent crowding of the points,

The third stage of the algorithm is to make the joint probability distribution of the data points in the low-dimensional space as similar as possible to the one in the high-dimensional space by using Kullback-Leiber divergence (KL divergence). KL divergence is a measure of the difference between two distributions.

$$D_{KL}(P \mid\mid Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad \text{(eq 3.6)}$$

The value ought to be smaller for distributions that are more similar to each other. The joint probability distribution for the data points in the low-dimensional space needs to be as similar as possible to the one in the original space, and this is achieved by using gradient descent. The cost function that the gradient descent tries to minimize is the KL divergence of the joint probability distribution P from the high-dimensional space and Q from the low-dimensional space.

$$C = KL(P \mid\mid Q) = \sum_i \sum_j p_{ij}\, log \frac{p_{ij}}{q_{ij}} \quad \text{(eq 3.7)}$$

The model accepts several hyperparameters that can be tweaked accordingly. Some of these parameters are related to the gradient descent, such as learning rate and the number of iterations. Another important parameter is perplexity. This hyperparameter is used for choosing the standard deviation in the Gaussian distribution representing the conditional probability distribution in the high-dimensional space. This parameter can be interpreted as the number of neighbours of a single data point.

However, t-SNE has some fallacies. By being a stochastic algorithm, it produces different results with every run. Additionally, despite preserving the local structure of the data, it might fail to preserve the global structure, which means that it can show clusters of data that are not really there.

4. METHODOLOGY

This section will function as a roadmap to the approach presented in this thesis and is composed of the following four subsections:

1. Data Collection, where the dataset is initially described.

2. Data Preprocessing, in which the process of transforming the raw input text and feature engineering is characterized.

3. Data Modelling, where the data partition, tuned hyperparameters and evaluation metrics that will be used are presented.

4. Data Analysis, where the results from the data preprocessing phase will be presented.

The overall organization of this section follows the schema presented in Figure 4.1.

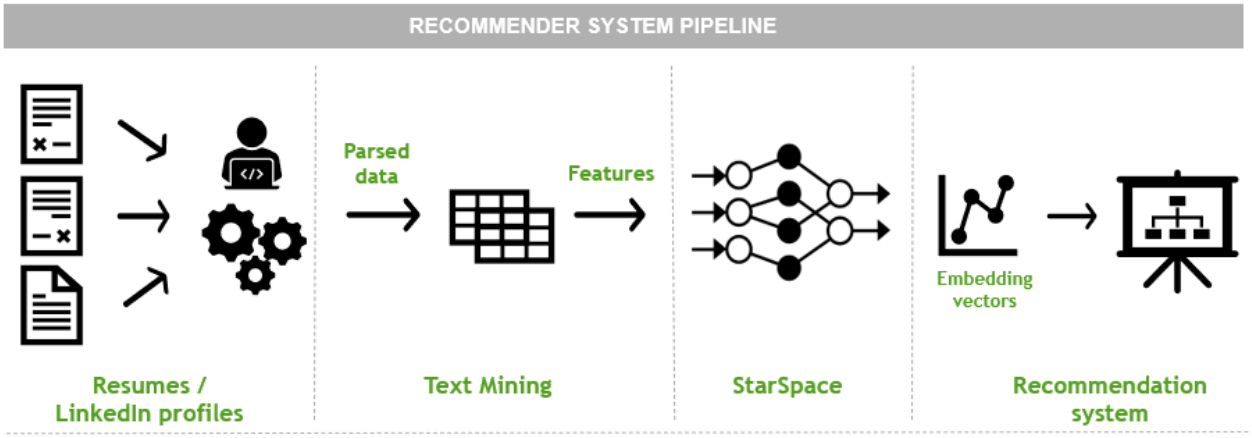


Figure 4.1 - Recommender system pipline

4.1. DATA COLLECTION

This subsection will portray and describe the dataset used for the assembling of the model. There are two datasets used in the proposed solution. The first dataset, from now on denoted as *resumes,* is completely unstructured and is consisted of 3,997 documents. These documents are people's resumes obtained from an IT consulting company as part of the recruitment process for jobs in the field of Business Intelligence, Data Science, Data Engineering, Software Maintenance, DevOps, Web Development, App Development, Graphic Design, Marketing, Management etc. The documents gathered were in three formats, pdf, doc and docx. The quantities of each of these formats are presented in Table 4.1.

Table 4.1 - Quantities of resumes per format

Format	Word
pdf	2,629
doc	1,285
docx	83

The second dataset, from now on denoted as *LinkedIn profiles*, is semi-structured and is consisted of 11,112 profiles scraped using a data-gathering tool – Phantombuster[1]. The process consisted of using two phantoms (tools) from Phantombuster – *LinkedIn Search Export*, for generating the URLs of the LinkedIn profiles according to applied filters and *LinkedIn Profile Scraper* for scraping the profiles provided. The filters applied on the search were by location, industry, language and service providers and the values inputted are presented in Table 4.2.

Table 4.2 - Filter applied in the LinkedIn search

Filter Type	Filter
Language	English
Location	Australia, North America, Western and Central Europe, Scandinavia
Industry	Information Technologies and Services, Computer Software, Computer Networking, Computer and Network Security
Service	IT Consulting, Consulting, Web Development, Application Development, Custom Software Development, Project Management, Mobile Application Development, Software Testing, Android Development, iOS Development, Graphic Design, Cybersecurity, Database Development, Software Testing and Data Reporting

[1] https://phantombuster.com/

The format in which the profiles are retrieved is a CSV file with a defined set of columns, indicating a feature from a user's LinkedIn profile. These features consist of personal information, location, job title, job description, job date range, company, school, school degree, school date range, skills etc.

4.2. DATA PREPROCESSING

This subsection will carry out a detailed characterization of all the steps performed in the preparation of the data for the model. The natural language, as a result of the human being, tends to be very random and unique. Computer algorithms do not deal very well with this randomness and thus require some normalization of the text before it is imputed in a machine learning model. The data preprocessing is focused mainly on normalizing the text inputs and cleaning the noise, which will reduce the variance and thus improve the overall model's performance. The process of text normalization consists of the following steps:

1. Filtering out resumes and LinkedIn profiles that are not written in English,
2. Removing capital letters by converting all text to lower case,
3. Removal of languages within the skills,
4. Acronym normalization,
5. Matching semantically same but syntactically different written skills,
6. Mapping fewer common skills to a more commonly present skill with similar meaning.

However, since there are two different datasets involved, one which is consisted of unstructured documents and another which is semi-structured scraped LinkedIn profiles, the data preprocessing for the first dataset has some specific data preprocessing steps included and those will be presented separately and initially, following the remaining transformations common for both. The features from the datasets that are considered for the problem apprehended are the skills containing a candidate's resume / LinkedIn profile.

Additionally, due to the nature of the problem tackled and the information that is aimed to be obtained from the data, a dictionary of LinkedIn skills is used in feature extraction in both datasets. Before describing the feature extraction from the datasets, this subsection will contain a characterization of the above-mentioned dictionary.

4.2.1. Language filter

The first step in the data preprocessing phase is to eliminate any input which is not in English in order to ensure that the word embeddings are not affected by the different languages in the datasets. Starting with the resumes, since they were gathered from a Portuguese consulting company, intuitively, a language filter had to be applied. The language of the resumes was detected using

langdetect[2], a python library ported from Google's language-detection. The results showed that only 30% of the resumes were in English. Table 4.3 shows the quantities for each language category.

Table 4.3 - Quantities of resumes per language

Language	Quantity
English	**1,168**
Portuguese	2,829

Likewise, similar analysis was conducted on the LinkedIn profiles. Despite applying English language as a filter on LinkedIn, the results retrieved showed that not all profiles scraped have this filter considered. Using the same method as before, around 7% of all the LinkedIn profiles were detected to not be written in English. The quantities of each are presented in Table 4.4.

Table 4.4 - LinkedIn profiles quantities by language

Language	English
English	**10,291**
Other	822

The size of each dataset after the filtering the language was 1,168 resumes and 10,291 LinkedIn profiles.

4.2.2. Skills dictionary

In order to be able to extract the skills from the resumes and LinkedIn profiles and differentiate an actual skill from a regular word, a dictionary of LinkedIn skills was gathered as an aid in the process. (Tabrizi, 2017) has provided a JSON file containing scraped data from the LinkedIn Topics Directory – a platform from LinkedIn providing exceptionally useful insights on skills, companies, universities and industries. Unfortunately, this platform has been discontinued, and thus the JSON file obtained from July 2017 is outdated. Nonetheless, its value has a significant contribution to the assessment of the problem apprehended.

The LinkedIn Topics Directory served as a dictionary of all the available skills on LinkedIn along with some additional metadata for the majority of them. The number of scraped LinkedIn skills is 33,188, and the metadata obtained from (Tabrizi, 2017) is the following:

- Number of people that have a specific skill,
- Top 10 companies where people who have a specific skill work,
- Top 10 similar skills which serve as keywords in search of a person with a specific skill,

[2] https://pypi.org/project/langdetect/

- Top 10 related skills, meaning people who have registered a specific skill also have registered these skills.

For the purposes of this work, only three of these features took part in the construction of the dictionary of LinkedIn skills: the name of the skill, the frequency of occurrence and the top related skills. Certain transformations were done in order to prepare the data for the intended usage.

Since the dataset available for the proposed work is not big enough, and the relationships between some skills could be lost, the idea was to generate those relationships by using the top related skills and their respective quantities from the above-mentioned dictionary and thus create a ratio for each related skill that will be later used as a weight on the inputs in the model. This feature was calculated as the ratio between the quantity of a top related skill and the quantity of the specific skill. This is done in order to ensure that the skills have some continuity into them, which means that, for example, if someone knows C++, it is highly likely that they know C as well. With this assumption, the skills are treated as dependent and correlated with each other. The given example is shown in Figure 4.2.


```
{
   "count":3715730,
   "skills":{
      "SQL":1195539,
      "JavaScript":1067628,
      "Microsoft Excel":1126591,
      "HTML":1380435,
      "Microsoft Word":918892,
      "Linux":939047,
      "C":2046070,
      "Java":1957788,
      "Microsoft Office":1591693,
      "Software Development":918171
   },
   "relations":{
      "sql":0.32,
      "javascript":0.29,
      "microsoft excel":0.3,
      "html":0.37,
      "microsoft word":0.25,
      "linux":0.25,
      "c":0.55,
      "java":0.53,
      "microsoft office":0.43,
      "software development":0.25
   },
   "name":"C++"
}
```


Figure 4.2 - Example relations of skill C++

The figure shows the metadata for the skill *C++,* which has 3,715,730 profiles that have it registered, its top skills and their respective quantities along with the derived relations. To give an interpretation, a ratio of 0.55 for related skill *C* means that from all the profiles that have *C++,* 55% of them have *C*. This value will represent the weight for each of the related skills in the model.

Furthermore, the analysis of the LinkedIn skills showed that there were some skills that were duplicates, which was noticeable when some of those skills show slight differences in the name, most often a character such as whitespace or a dash. Likewise, another analysis conducted was to detect skills that had slightly different relations but had the same name when trimmed. The merge of the relations is done in such a way that all the skills in the relations are considered. Furthermore, if a related skill appears multiple times, the bigger relation value is considered. An example of each scenario along with the end result are presented in Table 4.5 and

Table 4.6.

Table 4.5 - Scenario 1

Skill 1 (considered)	Skill 2	Relations
cloud computing	cloud-computing	{'business development': 0.28, 'solution selling': 0.28, 'pre-sales': 0.22, 'data center': 0.25, 'software as a service (saas)': 0.32, 'integration': 0.29, 'enterprise software': 0.34, 'project management': 0.3, 'virtualization': 0.24, 'management': 0.4}

Table 4.6 - Scenario 2

Skill1 (considered)	Skill2	Relations Skill 1	Relations Skill 2	Relations (merged)
web design	webdesign	{'web development': 0.22, 'adobe photoshop': 0.37, 'adobe creative suite': 0.22, 'microsoft office': 0.23, 'adobe illustrator': 0.26, 'html': 0.2, 'indesign': 0.2, 'cascading style sheets (css)': 0.2, 'social media': 0.26, 'graphic design': 0.37}	{'web development': 1.0, 'adobe photoshop': 1.0, 'adobe creative suite': 1.0, 'microsoft office': 1.0, 'adobe illustrator': 1.0, 'html': 1.0, 'indesign': 1.0, 'cascading style sheets (css)': 1.0, 'social media': 1.0, 'graphic design': 1.0}	{'web development': 1.0, 'adobe photoshop': 1.0, 'adobe creative suite': 1.0, 'microsoft office': 1.0, 'adobe illustrator': 1.0, 'html': 1.0, 'indesign': 1.0, 'cascading style sheets (css)': 1.0, 'social media': 1.0, 'graphic design': 1.0}

As mentioned before, not all of the skills from the LinkedIn Topics Directory had metadata, meaning they only appeared within the relations of other skills. A total of 910 skills with missing metadata were extracted from the relations and placed as a skill in the final dictionary. Table 4.7 shows a small batch of these skills.

Table 4.7 - Skills with missing metadata

Skill
search engine optimization (seo)
search engine marketing (sem)
internet protocol suite (tcp/ip)
business-to-business (b2b)
react.js

This concludes all the transformations done with the provided file of scraped LinkedIn skills and thus the creation of the dictionary of skills that will be further used in the data preprocessing.

4.2.3. Resume parser

Dealing with raw documents implies that, in order to extract information, a parser needs to be built. The goal of the parser is to transform the raw documents to text and further process it by segmenting its content and performing feature extraction. The process of parsing the resumes is consisted of some specific steps:

1. Conversion of the word documents to a pdf format,
2. The transformation process of document to textual data,
3. Segmentation of sections

First and foremost, in order to have a unified dataset that can be processed at once, a decision was made to convert all the word documents to a PDF format. One of the reasons why PDF was chosen over Word is the wide availability and support of libraries in python for extracting text from PDF documents. The process of converting the word documents to PDF was consisted of two steps. The first step was to convert all the files with .doc extension to .docx extension, due to the library further used only being supported in Windows 2007 above. This was done through the Microsoft VBA Editor in Word. The second step was to convert all the .docx files to PDF using the python library comtypes[3]. After all of the above-mentioned steps were carried out, the dataset was unified and consisted only of PDF documents.

The next step in the process is to convert the raw document to textual data that can later be transformed. This task was carried out by using pdfminer[4], a python library serving as a text extraction tool for PDF documents. More specifically, the conversion from documents to textual data was performed with pdf2text, a command line tool for extracting text from PDFs. Before performing the extraction, all the PDF documents were converted to text documents, by altering the extension from pdf to txt. The pdf2text method was then applied on the previously converted text documents and thus all the text from the documents was extracted and stored into a pickle[5] file.

Once the task was concluded and all the resumes were converted to text, the next stage in the preprocessing phase was to detect the sections of each resume and split it accordingly so they can be later processed. This task is very important in a resume parser as the end goal is to extract the correct information from the desired sections. Several methods were considered for the section detection, all of which are previously mentioned in the state of the art. Given the circumstances of the available dataset, a decision was made to define a set of possible section titles for each section separately. The sets were built based on analysis over the various section titles available in the resumes. The possible sections defined for the purpose of the parser were: personal information, skills, education, experience, languages, publications, achievements, annexes, additional information and ambiguous. The ambiguous section would contain sections which were very rare and do not belong to any of the other sections. A dictionary was built containing all the above-mentioned sections and was later used in the detection algorithm. After converting all the text to lower case, the split of each resume was

[3] https://pypi.org/project/comtypes/
[4] https://pypi.org/project/pdfminer/
[5] https://docs.python.org/3/library/pickle.html

done by identifying the starting positions of each detected section title from the dictionary and thus storing it separately for further analysis. Additional cleaning was done in regards for line breaks, page breaks, tab spaces, typographical symbols as well as pdf specific characters which were not detected by the UTF-8 encoding.

After the resumes have been parsed and transformed to a semi-structured format, the next step in the process is to perform feature extraction, text normalization and cleaning the noise in the data.

4.2.4. Skills normalization

The process of skills normalization with all the transformations that contain it, are described in the following subsection and correspond to both datasets.

First and foremost, the normalization of the skills starts by removing capital letters and converting all text to lower case, since python is case sensitive and would make the feature extraction much harder. The next step in the process was to remove the non-skill words from the skills section. Despite having the skills sections extracted from the resumes and LinkedIn profiles as well, analysis showed that additional preprocessing had to be carried out on both datasets in order to normalize the skills and filter out text which was noise. One relevant information to mention regarding the LinkedIn profiles is that under the *skills & endorsements* section on a LinkedIn profile, there is an option to input a subsection of languages as well, despite this section existing separately. Likewise, some skills sections parsed from the resumes also had the languages contained within. Since this information is irrelevant for the proposed solution, a decision was made to remove it from each input. This was carried out by using a manually created file containing around 65 languages which are mainly from European descent or are spoken by a rather big population. By doing a simple match, this data was successfully removed and thus the section text cleaned.

Given the nature of the text that is being processed, people most often replace the full name of a skill with its acronym. An example of such acronyms would be: *NLP* for *Natural Language Processing*, *ETL* for *Extract Transform and Load*, *BI* for *Business Intelligence* and so on. Having said this, the next step in the process was to handle the normalization of these acronyms and thus associate them with the full name of the skill and vice versa in order to reduce the variance. The method of acronym normalization considered is detecting a short form of a skill, its long form and a long form – short form combined together. An example of each is given below.

1) *Short form*, for example *NLP*,

2) *Long form*, for example *Natural Language Processing* and

3) *Long form (short form)*, for example *Natural Language Processing (NLP)*

With this in mind, the solution for associating a skill to its acronym was constructed by combining the above-mentioned forms, first by matching 1) in 3) and second by matching 2) in 3). The resulting skill was the format under 3) as common ground for three. A mapping table was constructed which was further used to detect the corresponding acronyms and map them to their respective names.

The next step in the transformations was to match semantically same but syntactically different written skills. These transformations would cover typos, plurals and differently written skills. The process encompasses matching of two datasets of skills, the user inputted skills from the datasets and

the skills from the dictionary. The dictionary is considered as ground truth and as such would be the target of the possibly erroneous user inputted skills. The implementation would require a calculated similarity of each skill in the datasets in relation to all the others skills in the dictionary. Following the approach presented by (Deep, 2020) in the literature review section, several different hyperparameters were tested before obtaining the best result. In Table 4.8 are shown the hyperparameter values were used to obtain the best results.

Table 4.8 - Optimal hyperparameters for the text normalization algorithm

Hyperparameter	Value
1st CSR matrix	Dataframe of distinct skills from the datasets to be normalized
2nd CSR matrix	Skills dictionary
n-grams	2
Top-N	5
Threshold	0.84

In regards to the LinkedIn profiles, all these transformations lead to a decrease in the number of distinct skills present in the dataset, from 25,693 to 22,568. However, further analysis showed that 98% of the skills had an occurrence of less than 1.5% of the total amount of LinkedIn profiles. This implied that these skills were not represented well enough and were thus considered noise. After filtering out the dataset according to this condition, the number of distinct skills i.e., vocabulary size, was 464.

As for the resumes, the size of the dataset was decreased for 100 resumes for which the parser did not detect a *skills* section, resulting in 1,068 samples. Furthermore, the above-mentioned transformations led to a less significant decrease of 6,706 to 6,632 distinct skills. Additionally, 94% of the extracted skills had an occurrence of less than 6.5% of the total amount of resumes, which either means an erroneous extraction or an underrepresented skill and were as well eliminated from the dataset resulting in a vocabulary of 388 skills.

4.2.5. Model Input

After all the transformations are concluded, the final step is to not only represent each candidate with the skills enlisted in their profile but also with the related skills previously generated. However, since the related skills and their respective weights are obtained from a different dataset, i.e., all the LinkedIn profiles available at the moment of extraction, there are some additional skills that occur within them that do not occur at all in the datasets used in this work. Thus, to prevent expanding the universe of available skills in the datasets to unknown ones, the related skills considered in the end were the only ones that have occurred within the datasets. By doing this, the related skills serve their purpose of providing the model with the "missing" relationships between the skills and aid the learning process. Therefore, the original set of skills are all associated with a weight of 1, whilst the related ones will have the calculated weight as explained in subsection 4.2.2. Additionally, in order for the model to treat the skills as labels, as it is intended in this use case, the prefix *__label__* is appended before every skill. An example for a single input in the model, i.e., a single candidate is as follows: *__label__1 __label__2 … __label__M*.

4.3. DATA MODELING

The data modelling succeeds the preprocessing stage and is consisted of the characterization of the chosen train mode in StarSpace, the partitioning of the data into train, test and validation datasets, as well as the hyperparameters tweaked in the process. Thus, this subsection will consist of a) description of how the chosen train mode works, b) data partitioning and c) definition and reasoning behind the chosen hyperparameters in training.

4.3.1. Content-based recommendation (trainMode=1)

As described in the literature review, StarSpace can be used in different training modes, depending on the use case. The proposed solution uses train mode 1, as the aim is to build a recommender system as an end result. Considering the context of the datasets, in the chosen train mode, a user is represented as a bag-of-skills which are enlisted in their resume or LinkedIn profile. The way the model works is that it does not learn direct embeddings of the users. Rather a user will have an embedding which is the average of all the skills embeddings the user possesses. The skills, on the other hand, are embedded directly as features in the dictionary. This use case gives better results when the number of users is bigger than the number of skills, and the number of skills for every user is small on average. Each input is a single user represented by the bag-of-skills, where each skill is treated as a featured label. Additionally, each of the skills is assigned the weight computed in the data preprocessing step.

As each input is represented as a collection of labels, a positive entity pair (a, b) is generated such that b is a randomly selected label from the collection, while the rest of the labels from the collection are selected as a. The positive entity pair represents a correct output, or in the given context, set of skills that are most likely to occur together. The negative entities b_i^- are generated by using the negative sampling method proposed by (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). StarSpace is trained to map similar features in closer proximity in the vector space rather than the negatively sampled features. An illustrative example of a set of entity pairs given the context of this work would be:

$$\big((python, machine\ learning, pandas, data\ science, NLP), neural\ networks\big)$$

$$\big((python, machine\ learning, pandas, data\ science, NLP), node.js\big)$$

$$\big((python, machine\ learning, pandas, data\ science, NLP), web\ development\big)$$

$$\big((python, machine\ learning, pandas, data\ science, NLP), photoshop\big),$$

In which the subset of *python, machine learning, pandas, data science* and *NLP* and considered as a, while *neural networks,* a *true* label, is considered as b. Moreover, the b_i^- entities are accordingly the skills *node.js, web development* and *photoshop*, which are considered negative as they are not closely related to the previous.

4.3.2. Data Partition

The objective of this task is to describe the partitioning of the data into a training, test and validation dataset. The initial datasets are respectively split, by random, into partitions of 75% training, 10% validation and 15% test dataset. These respective datasets are later used as inputs in the model, each of which is utilized for a specific part of the process.

4.3.3. Hyperparameters

Several hyper parameters were tweaked in order to test their impact on the overall performance of the model. A grid search algorithm was implemented in order to determine the optimal values for each of the hyperparameters in an elegant manner. The hyperparameters that were chosen for the grid search are described in below.

Dimensionality – size of the embedding vectors. The dimensionality of word embeddings has an influence on its performance (Yin & Shen, 2018). Smaller embedding vectors would mean compressing the words too much and not representing the semantics well enough. On the other hand, too large vectors would require the model to learn more parameters which will thus require more data. Moreover, a larger dimensionality would mean that the model would not only consider relevant information, but also consider noise while learning the embeddings. The default value for this parameter is 100.

Epochs – number of epochs, hence the number of times the whole dataset will pass through the model. The number of epochs, as any other hyperparameter is determined through a matter of trial and error. Depending on the type of problem assessed by the model, a very large number of epochs could lead to an overfitting of the model, as it will learn the dataset to an extent that it won't be able to generalize well enough. Conversely, a small number of epochs could lead to an underfitted model which occurs when the model is unable to capture the underlying patterns in the data. An epoch is comprised of one or more batches. The default value for this parameter is 5.

Batch size – size of mini batch in training. A batch is the number of examples used in each iteration of the training phase. The smaller the batch size the higher the variance, meaning that the model will update its parameters more frequently, leading to big oscillations in the loss function and a slower overall convergence to optimal embeddings. However, by introducing a very high batch size, the rate of the updates of the parameters will be much lower, leading to an averaging of the whole batch of examples, which might vary a lot from one to another, thus introducing bias to the model. The batch size is a typical example of the bias vs variance trade-off. The default value for this parameter is 5.

Validation patience – the number of iterations of validation where the model does not improve before the training is stopped. An iteration is the number of batches needed to complete an epoch. This hyperparameter is crucial for maintaining the optimal hyperparameter values and stop the model from overfitting. A smaller value might be insufficient and lead the model to a higher validation error, while a larger value might lead to an overfitted model, which is exactly what is aimed to avoided with the use of this parameter. The default value for this parameter is 10.

Negative Search Limit – number of negatives sampled during each batch. Too few negatives sampled would lead the model to not be able to differentiate well enough between the positive and negative samples. Too many negatives sampled would introduce too much noise in the model. The default value for this parameter is 50.

Table 4.9 presents the variations of hyperparameters tested in the process of obtaining the optimal result.

Table 4.9 - Set of values for hyperparameters tuning

Hyperparameter	StarSpace API	Values
Dimensions	dim	[50, 150, 250]
Epochs	epoch	[10, 30, 50]
Negative Samples	negSearchLimit	[50,100,150]
Batch Size	batchSize	[5,10]
Early Stopping	validationPaticence	[10,15,20]

4.3.4. Evaluation metrics

StarSpace has built-in metrics for evaluating the performance of the model. Aside from metrics for validating the results, such as hits @ k and mean predicted rank, the model provides the standard train and validation loss and error as a metric for evaluating the model in the training phase.

4.3.4.1. Error

The error function is computed as the average loss of all the examples in all the mini-batches of a single epoch. The loss function is updated for every mini batch within an epoch.

4.3.4.2. Hits @ k

The hits@k metric is the equivalent of precision@k metric for text classification. Precision is defined as:

$$precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad \text{(eq 4.1)}$$

which can be interpreted as the proportion of the retrieved entities that are relevant to the user. In a binary classification, precision takes all the entities into consideration, however precision@k only evaluates the top k results retrieved by the system. This variant of the metric does not consider the order of retrieved entities but rather the percentage of relevant entities retrieved in the top k results.

4.3.4.3. Mean Rank

The mean rank metric represents the mean predicted rank of the chosen entity among n entities. This metric represents the arithmetic average of the positions of the entities ranked ascendingly and measures the ability of a retrieval system to score a *true* result among all possible results.

4.4. DATA ANALYSIS

This subsection will carry out descriptive statistics based on the analysis performed over both datasets before and after the data preprocessing. Considering the type of data at disposal, the analysis will cover a general overview of the data and present measures such as frequency counts of the skills available in the datasets and average number of skills per candidate. All of the graphs will be shown for both datasets, side by side. First and foremost, in order to obtain a general idea of the skillset of the users, an analysis was done of the most frequent skills. However, since every candidate in the model is represented with the original set of skills and the related skills, the following presented will show the top 20 most frequent skills before and after merging both skillsets.

Figure 4.3 - Top 20 most frequent skills (Original) – LinkedIn Profiles

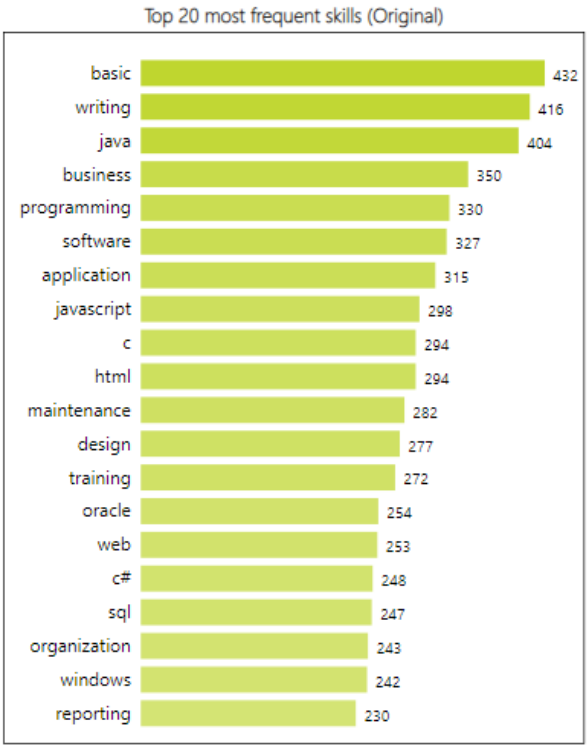


Figure 4.4 – Top 20 most frequent skills (Original) – Resumes

At first glance, one can conclude that the majority of the skills in the LinkedIn profiles are non-technical, an umbrella group that encompasses areas such as management, entrepreneurship or sales. This can be confirmed by the fact that only four skills from the top 20 are actually closely related to the area of software development.

However, it is harder to make such a conclusion for the skills obtained from the resumes, as there are some which presumably imply erroneous parsing like *business*, *software*, *application, writing,* etc. As described in 4.2.2, the dictionary used in the feature extraction process contains various different skills available from the LinkedIn database, thus leading the regex methods applied to possible wrongful matches. Nevertheless, the most frequent skills from the resumes imply that the candidates come from a more technical background.

Figure 4.5 - Top 20 most frequent skills (Original + Related) – LinkedIn profiles

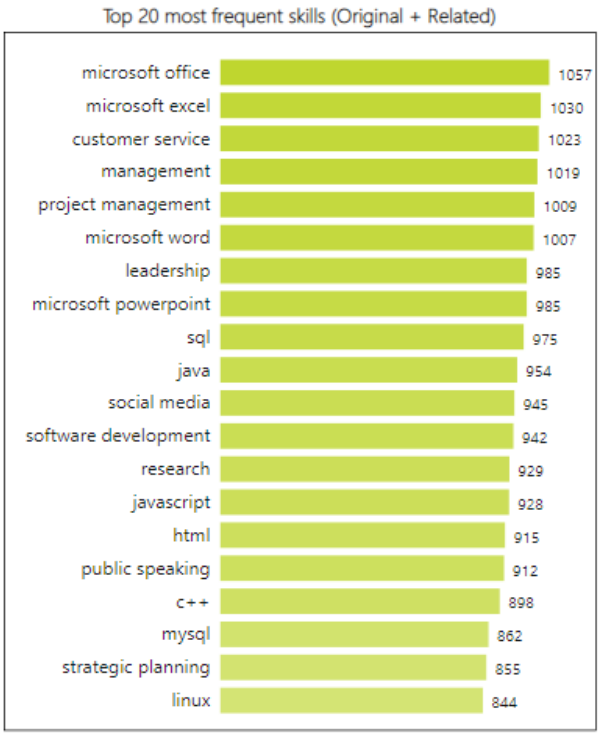
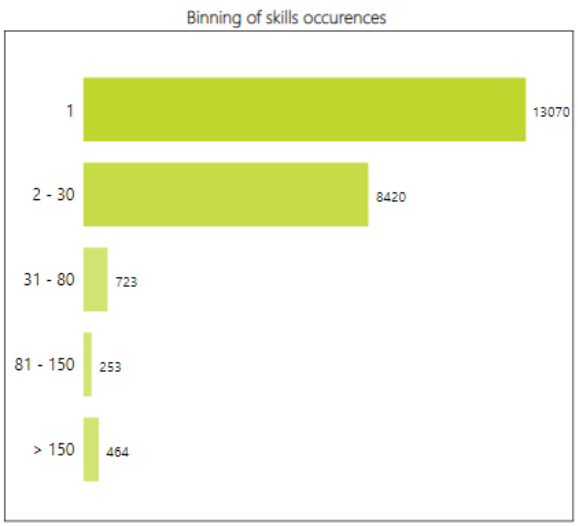


Figure 4.6 - Top 20 most frequent skills (Original + Related) – Resumes

It can be observed that both lists differ with the previous to some extent and that is mainly due to the related skills that are appended to each candidate's original skillset. Some of these skills, like the Microsoft Office Suite, are generic, in the sense that most people have them enlisted in their profiles and logically they appear in the related skills very often.

Furthermore, the data exploration process continues with computing the occurrences and the number of skills in order to identify how well each feature is represented in the datasets. What the graph below represents is the number of skills that occur within the given range of values, for instance, in the LinkedIn profiles there are 13,070 skills that occur only once, 8,420 that occur from 2 to 30 times and so on.



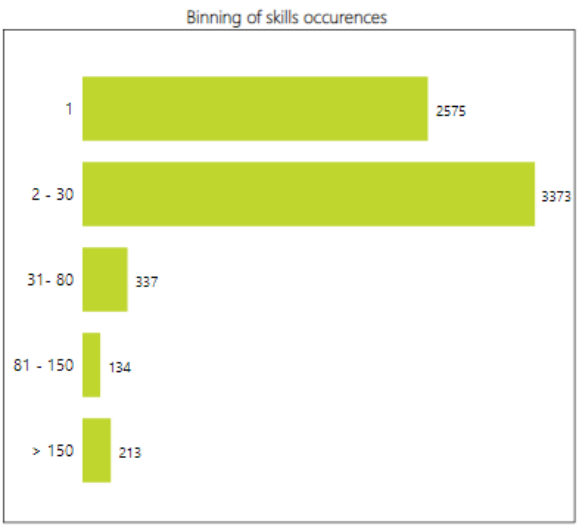


Figure 4.7 - Binning of skills occurrences – LinkedIn profiles

Figure 4.8 - Binning of skills occurrences - Resumes

This particular analysis served to filter the vocabulary size, which lead to a reduction in its variance. The assumption made was to remove skills which are not well represented. Several different thresholds were applied in order to obtain the best result possible. Another interesting statistic explored is the number of skills each user is represented with. Despite the fact that there is no evidence that the dimensionality of the word vectors is correlated with the vocabulary size, it served as a good starting point in the tuning of this particular hyperparameter.

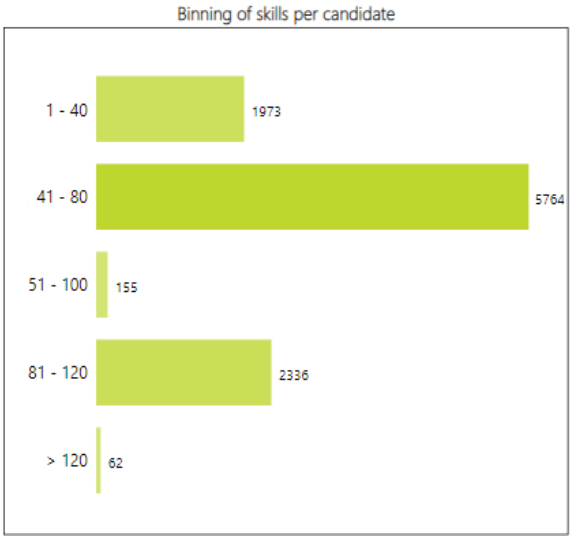


Figure 4.8 - Binning of skills per candidate – LinkedIn profiles

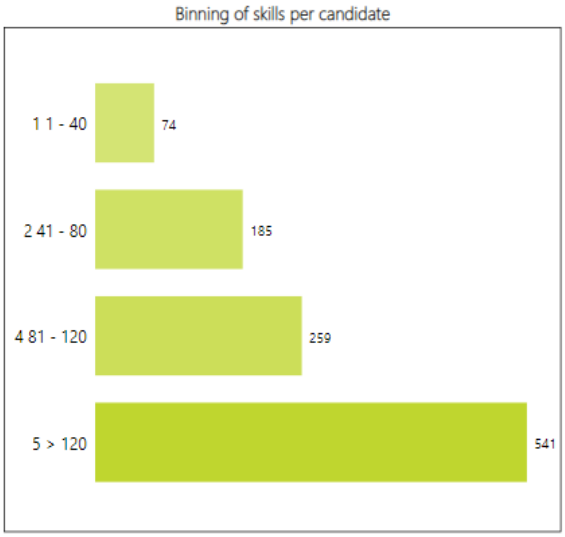


Figure 4.9 - Binning of skills per candidate – Resumes

The average number of skills per candidate in the LinkedIn profiles is 74, while for the resumes this number is 125. This large number is a result from the related skills attributed to the original set. However, the larger average number of skills per resume is presumablly due to erroneous feature extraction.

5. RESULTS AND DISCUSSION

This section is the pinnacle of this work and presents the results of the proposed methodology along with a discussion regarding the experiments that lead to the optimal hyperparameters chosen for the model. It is comprised of a brief summary of the model performance metrics together with its embeddings, outputs and several practical examples of most adequate candidates' retrieval for given job positions. Additionally, multiple conducted experiments will be showcased, that not only validate the assumptions made throughout this work, but also illustrate the importance of certain hyperparameter choices.

5.1. RESULTS

5.1.1. Training performance

During the training phase, the model is tuned in such way that it learns how to optimize the loss function in regards to the training set. At the end of every iteration, it evaluates its performance over an independent set of data, known as the validation set, in order to ensure that the model does not overfit. However, these quantitative metrics used to assess the model's performance are not a clear indicator of the quality of the embeddings produced as an output. The loss function will be used to assess the training performance, whereas for the evaluation, hits@k and Mean Rank will be presented. The results shown are regarding the optimal hyperparameters which are given in Table 5.1.

Table 5.1 - Optimal hyperparameters for StarSpace

Hyperparameter	Value
Dimensions	150
Epochs	30
Negative samples	50
Batch size	5
Early stopping	20

The model was run with the same hyperparameters on both datasets, the LinkedIn profiles and the resumes. The results are shown in Figure 5.1 and Figure 5.2 correspondingly.

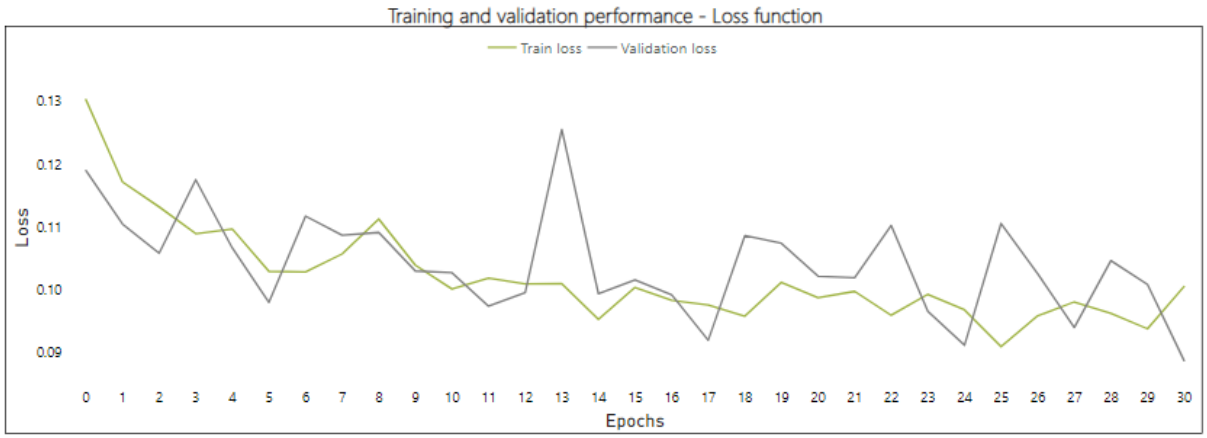


Figure 5.1 - Training and validation sets performance – Loss function – LinkedIn profiles

When observing the performance of the model on the dataset of the LinkedIn profiles, it can be noticed that the loss function on both training and validation sets converges, although there is some noticeable variance on the validation set. Early stopping od 20 iterations is used as a regularization method to avoid overfitting of the model.

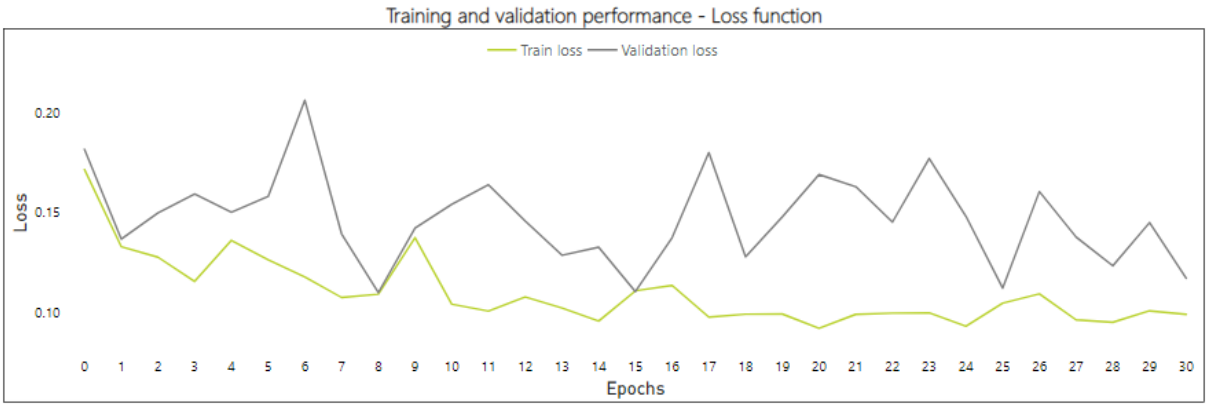


Figure 5.2 - Training and validation sets performance – Loss function – Resumes

However, when running the model on the dataset of the resumes, the results show inferior performance. The validation loss is much higher and does not seem to converge throughout epochs, meaning that it is not able to generalize well. The fact that the dataset is completely unstructured and the parser is not as robust contributes to these results, in addition to the smaller amount of data inputs available for the model to learn from.

5.1.2. Skills embeddings

As referred to in section 4.4, certain experiments were done by reducing the vocabulary size. Even after all the transformation steps applied, the number of distinct skills was more than double of the dataset size and a set of them had very few occurrences. Intuitively, an assumption can be made that the model would not only suffer lack of data given the complexity of the vocabulary size, but also receive a lot of noise and thus not be able to learn the embeddings correctly. Having said this, certain skills that had very small occurrences throughout the whole dataset were filtered out. Initially, for the dataset of LinkedIn profiles, the threshold applied was an occurrence of above 1.5% of all the available inputs, resulting in a vocabulary size of 464 skills. Like-wise, for the resumes this threshold was slightly higher due to the small ratio of the vocabulary size and the data inputs, thus an occurrence of less than 6.5% of all the available inputs was only considered, resulting in a vocabulary size of 388 skills.

The skills embeddings produced by the model for the dataset of LinkedIn profiles are shown in Figure 5.3.

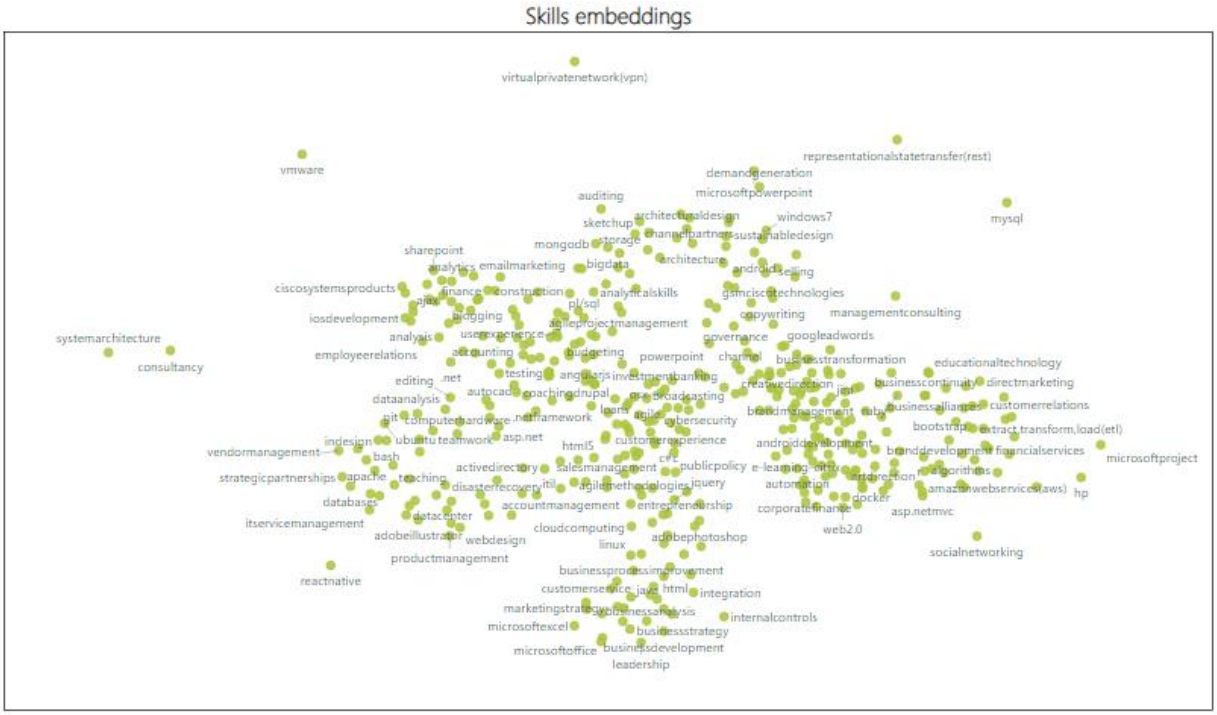


Figure 5.3 - tSNE output of the skills embeddings

One can observe that StarSpace does not provide clusters as blobs, but rather arms of a star. When zooming into the arms of the star to see more clearly how the skills were grouped, despite some related skills being in close proximity in the vector space, other ones were not as coherent. For instance, Figure 5.4 and Figure 5.5 represent the embeddings from two of the arms of the star.

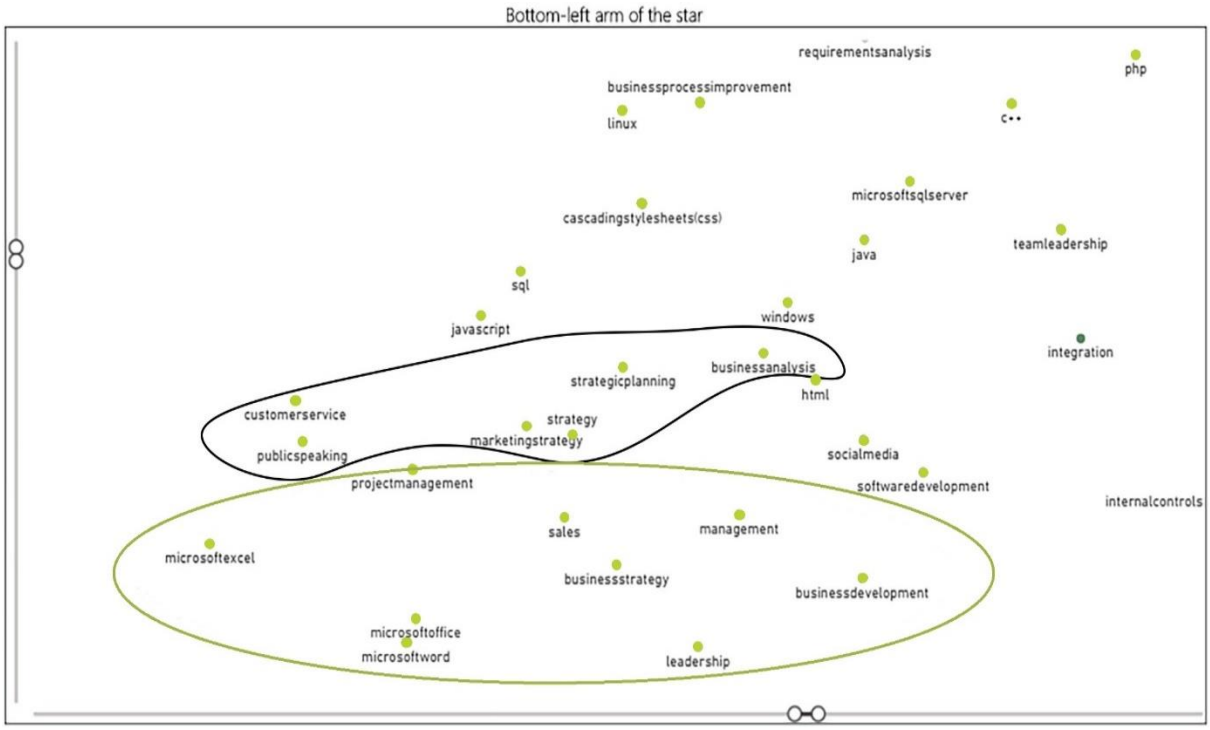


Figure 5.4 - Skills embeddings in the bottom-left arm of the star

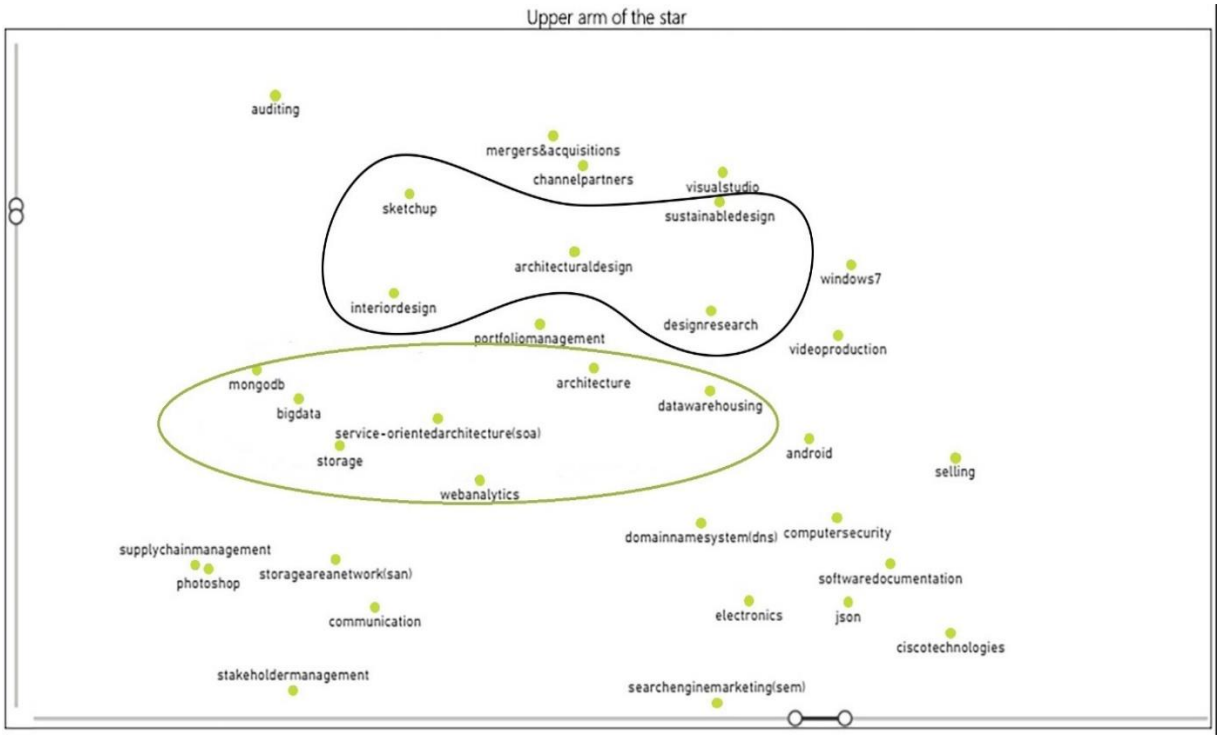


Figure 5.5 - Skills embeddings in the upper arm of the star

However, since t-SNE is a heuristic algorithm with a convex cost function, it is prone to produce different outputs with every initialization. Additionally, it is highly dependent on the hyperparameters chosen, one of which being the perplexity. Therefore, these outputs are not consistent and thus unreliable in order to properly evaluate the quality of the embeddings. An alternative approach would be to generate the nearest neighbors for a set of the embeddings and validate if they are closely related in terms of a job position. Having said this, the skills embeddings for the resumes will be presented only by the nearest neighbors approach.

The examples and their corresponding 10 nearest neighbors from the dataset of LinkedIn profiles are presented in Table 5.2, where the *Skill* column contains the examples for which this evaluation was performed, i.e., the skills, and nearest neighbors are the embeddings in their closest proximity, along with the calculated degree of similarity. For purposes of presentation, the label prefix is omitted.

Table 5.2 - Nearest neighbors for the skills embeddings – LinkedIn profiles

Skill	Nearest neighbors	Similarity
Python	Linux	0.722398
	C	0.670724
	Java	0.623791
	Javascript	0.580158
	C++	0.573350
	MySQL	0.521145
	SQL	0.506378
	C#	0.443652
	Bash	0.407018

	Windows	0.371283
Project Management	Leadership	0.804328
	Management	0.795947
	Microsoft Office	0.71733
	Microsoft Excel	0.689631
	Strategic Planning	0.625083
	Microsoft Word	0.611664
	Business Development	0.596941
	Change Management	0.548703
	Marketing	0.545072
	Customer Service	0.544908
Graphic Design	Adobe Creative Suite	0.837977
	Adobe Illustrator	0.823914
	Web Design	0.819836
	Indesign	0.773138
	Adobe Photoshop	0.478329
	Photography	0.412290
	Blogging	0.255544
	Socialmedia	0.251899
	Wordpress	0.244313
	Advertising	0.234385
C++	SQL	0.78949
	C	0.780733
	Java	0.723974
	C#	0.692142
	XML	0.649427
	Linux	0.588765
	Python	0.573350
	Javascript	0.552689
	HTML	0.540757
	MySQL	0.515474
Marketing	Social Media	0.734581
	Online Marketing	0.699936
	Social Media Marketing	0.694485
	Business Development	0.679288
	Marketing Strategy	0.651426
	Online Advertising	0.646929
	Strategic Planning	0.560859
	Business Strategy	0.558255
	Leadership	0.551333
	E-commerce	0.543282
HTML	PHP	0.729689
	Javascript	0.673449
	Java	0.646176

	Cascading Style Sheets(Css)	0.626763
	MySQL	0.622788
	C	0.595573
	Adobe Photoshop	0.588608
	Jquery	0.587062
	C++	0.540757
	C#	0.484910

When observing the nearest neighbors for the skill *marketing,* it can be assumed that the majority of these skills correspond to the field itself. Since this specific skill can also be interpreted as an area of expertise, it can be noticed that some of its neighboring labels are actually subfields of marketing, such as *social media marketing*, *online advertising* and *online marketing*. Others such as *marketing strategy* and *strategic planning* are expertise which are closely related with the field. Moreover, some of the skills such as *business strategy* or *business development* are competencies which border with business development as a field, but are nonetheless used interchangeably.

A more challenging example would be a multidisciplinary skill like *python.* Python as a coding language spans throughout several subfields of IT such as Data Science, Web Development or DevOps, making it harder for the model to correctly embed it in the vector space. Some of its neighboring labels are other programming languages like *C, C++, C#* or *Java* which although different, are very common among IT candidates because they are often included as part of their skillset. Additionally, most Python applications are developed in *Linux*, with *bash* being the command language for executing scripts, which in fact validates their close proximity in the vector space. A label such as JavaScript is more common for an area like Web Development, while SQL and MySQL are skills that could be more associated with Data Science, as they are both querying languages used for analyzing databases.

As a matter of comparison of the two models, the nearest neighbors for the same examples were obtained for the dataset of resumes.

Table 5.3 - Nearest neighbors for the skills embeddings – Resumes

Skill	Nearest neighbors	Similarity
Python	Shell Scripting	0.412858
	Routing	0.394275
	SQL	0.382964
	Twitter	0.318225
	Construction	0.313599
	Telecom	0.298699
	Machine Learning	0.297703
	Events	0.286437
	Continuous Improvement	0.273059
	Chemistry	0.272843
Project Management	Customer Service	0.554446
	Graphic Design	0.509754
	Leadership	0.494189

	Software Documentation	0.491384
	Service - Oriented Architecture (SOA)	0.464253
	Microsoft Office	0.401446
	Investments	0.401319
	Ecology	0.390912
	Digital Marketing	0.386764
	Negotiation	0.372855
Graphic Design	Adobe Photoshop	0.66633
	Adobe Illustrator	0.611562
	Leadership	0.531271
	Project Management	0.509754
	Wireless Technologies	0.462757
	Digital Marketing	0.431325
	Product Management	0.420257
	Program Management	0.417645
	Microsoft Excel	0.413010
	Adobe Creative Suite	0.412126
C++	C	0.856979
	Java	0.710362
	MySQL	0.672454
	Javascript	0.616220
	Microsoft Word	0.572703
	Software Development	0.509616
	Research	0.493199
	Linux	0.482972
	PHP	0.479872
	SQL	0.452620
Marketing	Marketing Strategy	0.576612
	Social Media	0.545157
	Business Strategy	0.539996
	Strategy	0.486134
	Social Media Marketing	0.457117
	Strategic Planning	0.454019
	Video Production	0.436849
	Microsoft Powerpoint	0.434614
	Leadership	0.425067
	Investments	0.419977
HTML	Web Development	0.530001
	Javascript	0.501092
	Unix	0.497654
	Perl	0.443475
	Electrical Engineering	0.441803
	Shell Scripting	0.426458
	Litigation	0.413693

	Operations Management	0.394454
	Wordpress	0.383370
	.NET	0.381892

As can be observed, the nearest neighbours obtained for the resumes are not as adequate as the previous. Although some skills appear to be related in practice, for instance, the skills for *marketing* or *c++,* other ones such as *python* or *project management* are not as coherent with the skills in their nearest proximity. The lack of data undoubtedly affects how well the model will learn the embeddings of the skills as it has much fewer examples from each skill in order to correctly place it in the vector space.

5.1.3. User embeddings

After the skills embeddings are generated, the next step is to create user embeddings based on the skills each candidate is represented with. This is carried out by computing a weighted average over the produced skills embeddings and their corresponding weights. A weighted average is done in order to ensure that the related skills which have a lower weight do not contribute equally to the resulting vector, as opposed to the highly related ones, which may or may not be a part of the original skillset of a candidate.

Given a candidate C represented with a set of skills S, the corresponding user embedding E_C is calculated as:

$$E_C = \sum_{i=1}^{N} \frac{1}{N} w_i s_i \,, \qquad \text{(eq 5.1)}$$

where N is the number of skills and $w_i \in W$ are the weights associated with each skill $s_i \in S$. An example is illustrated in Table 5.4 where each of the skills embeddings is represented with only three dimensions.

Table 5.4 - Example of a user embedding

Skills	Embeddings	User Embedding
__label__python:1	$[0.1 \quad 0.3 \quad 0.5]$	$\frac{1}{3}\begin{bmatrix} 1 \times 0.1 & 1 \times 0.3 & 1 \times 0.5 \\ 0.9 \times 0.6 & 0.9 \times 0.8 & 0.9 \times 0.2 \\ 0.8 \times 0.9 & 0.8 \times 0.1 & 0.8 \times 0.5 \end{bmatrix} = [0.45 \quad 0.36 \quad 0.36]$
__label__pandas:0.9	$[0.6 \quad 0.8 \quad 0.2]$	
__label__sql:0.8	$[0.9 \quad 0.1 \quad 0.5]$	

Given the poor results on the dataset of resumes previously described in 5.1.2, this subsection will only characterize the user embeddings produced from the dataset of LinkedIn profiles. The vector space of the user embeddings for the dataset of LinkedIn profiles is presented in Figure 5.6.

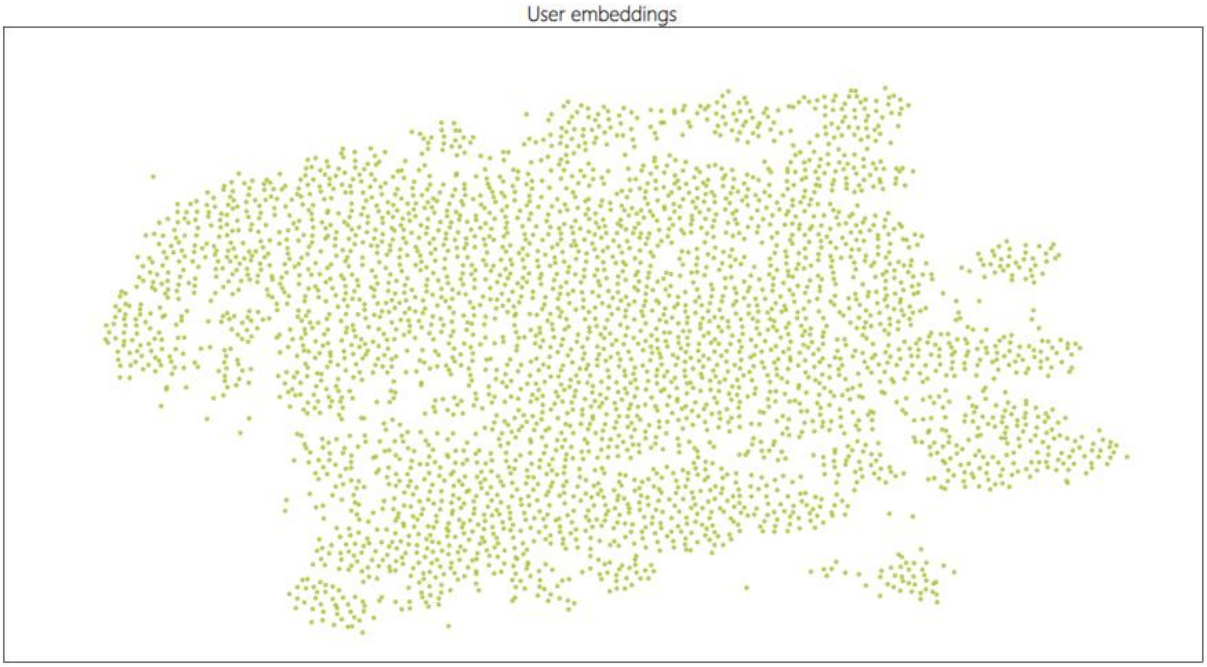


Figure 5.6 - tSNE output of the user embeddings

Likewise, due to the unreliability of t-SNE and a large number of data points, the quality of the embeddings is validated by the nearest neighbours approach, where the input is a set of skills which represent a given job position. A job position is represented in the same way as a candidate, hence as a vector of all the skills required. In case weights are not provided, the model assumes the default weight of 1 for each given skill. Having said this, a job position would represent the perfect candidate as it would include all the required skills. Thus the candidates in its closest proximity would be the adequate ones for the job position. The examples illustrated in Table 5.5, Table 5.6 and Table 5.7 are based on real job advertisements obtained from LinkedIn, where each job position is broken down into a set of required skills enlisted in the advertisement itself. However, the vector of the job position will only consist of skills that belong to the set used to train the model, disregarding any other requirements such as education, languages, experience, seniority etc. The following three distinct job positions were chosen to evaluate the quality of the embeddings: 1) Data Scientist, 2) Marketing Specialist, and 3) Front-end developer. For further reference, the full descriptions of the job advertisements are available in appendix 9.2.

The first job advertisement is for the position of a Data Scientist at a consulting company. The company is offering a full-time job for an associate-level Data Scientist with a degree in Computer Engineering and experience with non-relational databases and technologies such as R, Python, Java or Scala. Additionally, it is required that the candidate is fluent in English, has good communication skills and is flexible and dynamic. The input vector for the job position consists of *Python, Machine Learning, Data Science, Java, R* and *Databases*. The top three candidates obtained by the nearest neighbours algorithm are displayed in Table 5.5.

Table 5.5 - Top 3 candidates for the job position of a Data Scientist

Job Position	Candidates		
Data Scientist Python, Machine Learning, Data Science, Java, R, Databases	**Candidate 1**	Current Job Position	CEO | Chief Data Strategist @ a company that provides data science solutions
		Top 10 Skills	1. Algorithms 2. Machine Learning 3. Mathematical Modeling 4. Data Analysis 5. Data Mining 6. Computer Science 7. Python 8. Big Data 9. Artificial Intelligence 10. C++
	Candidate 2	Current Job Position	CEO @ a financial analytics AI company
		Top 10 Skills	1. Machine Learning 2. R 3. Data Analysis 4. Deep Learning 5. C++ 6. Python 7. Data Science 8. SQL 9. MATLAB 10. TensorFlow
	Candidate 3	Current Job Position	CTO @ an IT company
		Top 10 Skills	1. R 2. TensorFlow 3. Python 4. Deep Learning 5. Software Development 6. Software Architecture 7. Computer Vision 8. SQL 9. Data Engineering 10. Mathematical Physics

As can be observed, all three candidates are executives of a C-level job position, two Chief Executive Officers and one Chief Technology Officer. Although it is not precisely known how balanced the dataset is in regards to the level of seniority of the candidates, one might argue that the model retrieves candidates who are more experienced. Therefore it is more probable that they will have the skills the job requires, as well as other ones which are highly related to the former. For instance, skills such as *Machine Learning, TensorFlow, Deep Learning, Computer Vision, Data Science, Data Analysis, Data Mining* etc. are clearly skills that are linked to a Data Scientist. Additionally, *Python* and *R* are a perfect match to the given requirements. However, despite the fact that none of the top 3 candidates have Java or Scala as a skill in their profile, it can be observed that some do have C++ , also an object-oriented language, which has a high weight of 0.53 for Java in its related skills in the dictionary. Moreover, Java

is highly related with Scala with a weight of 0.79. Likewise, none non-relational databases appear in these candidates' profiles, however some have enlisted Big Data and Data Engineering, which are the subfields to which non-relational databases belong. Figure 5.7 illustrates a visual representation of the skills distribution of these candidates.

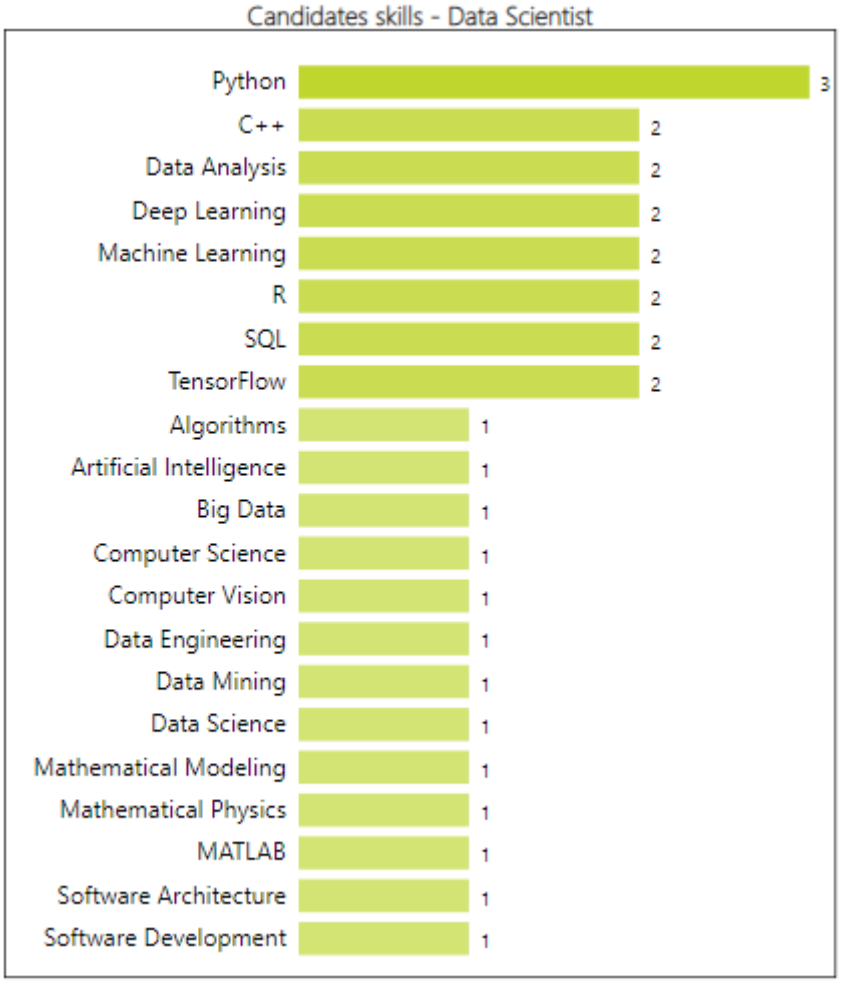


Figure 5.7 - Skill counts for the top 3 candidates for the Data Scientist job position

The second job advertisement is regarding the position of a Marketing Specialist in a Marketing Agency. The employer is offering a full-time job to an associate level Marketing Specialist with an expertise in development and optimization of Search Engine Marketing as well as Social Media Marketing campaigns. The job requires extensive knowledge in Google Ads, Facebook Ads, Google Analytics, good knowledge in Web Marketing and knowledge in JavaScript. The top three candidates obtained for the given job position are illustrated in Table 5.6.

Table 5.6 - Top 3 candidates for the job position of a Marketing Specialist

Job Position	**Candidates**		
Marketing Specialist	**Candidate 1**	Current Job Position	Search Engine Optimization Manager @ an internet marketing service company

Google Ads, Google Analytics, Facebook, Search Engine Optimization (SEO), Search Engine Marketing (SEM), JavaScript, Web Marketing, Social Media Marketing		Top 10 Skills	1. Search Engine Optimization (SEO) 2. Search Engine Marketing (SEM) 3. Social Media Marketing 4. Social Media Optimization (SMO) 5. Google AdWords 6. Google Analytics 7. Content Management 8. Web Marketing 9. Web Development 10. Facebook
	Candidate 2	Current Job Position	Chief Technology Officer @ a marketing company
		Top 10 Skills	1. Google Analytics 2. Search Engine Optimization (SEO) 3. Search Engine Marketing (SEM) 4. Web Marketing 5. Social Media Marketing 6. Google AdWords 7. Web Analytics 8. Online Marketing 9. CMS 10. WordPress
	Candidate 3	Current Job Position	Marketing Technology and Business Development Manager @ a law firm
		Top 10 Skills	1. Customer Relationship Management (CRM) 2. Email Marketing 3. User Experience 4. Content Marketing 5. Social Media 6. Marketing Strategy 7. User Interface 8. Web Marketing 9. Web Analytics 10. Search Engine Optimization (SEO)

In a similar manner, the candidates retrieved are on a higher hierarchical position, two managers and one Chief Technology Officer. The skillset of these candidates is an almost perfect match in regards to the job position vector, as the majority of the required skills are either directly or indirectly linked to the candidates. The only required skill which is not enlisted in either of the candidates' profiles is JavaScript. However, some of the candidates have skills such as CMS or Web Development which are both associated with JavaScript in the dictionary with a weight of 0.9 and 0.27 accordingly. This is a

clear example of the importance of the relations incorporated in the model. The visual representation of the distribution of skills for these candidates is shown in Figure 5.8.

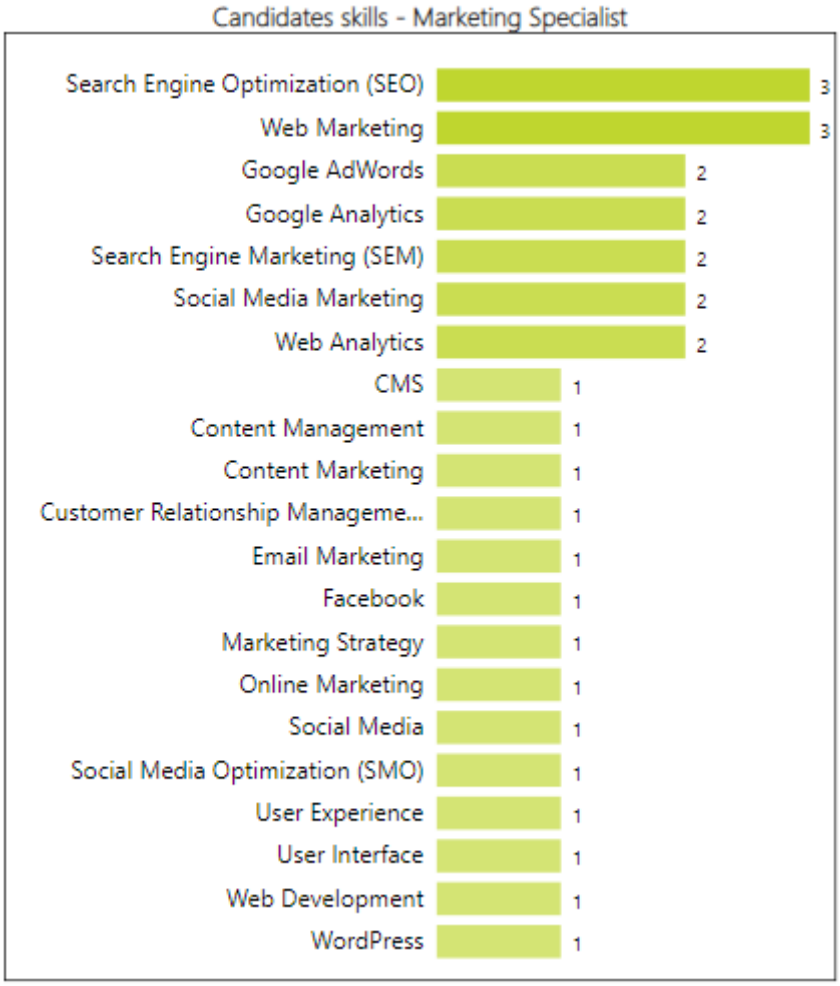


Figure 5.8 - Skill counts for the top 3 candidates for the Marketing Specialist job position

Lastly, the user embeddings were evaluated for the job position of a Front-end Developer. The job position described in the advertisement is fit for an experienced Front-end Developer that has strong computer science fundamentals and is fluent in technologies such as React.js, Node.js or JavaScript. Additionally, the candidate ought to have extensive experience in software design and development, as well as be able to work in an agile engineering environment. The candidates that scored best are presented in Table 5.7.

Table 5.7 - Top 3 candidates for the job position of a Front-end Developer

Job Position	**Candidates**		
Front-end developer JavaScript, Node.js, React.js, Front-end development	**Candidate 1**	Current Job Position	Software Engineer @ an IT company
		Top 10 Skills	1. JavaScript 2. Node.js 3. Computer Science 4. Object Oriented Design 5. Web Development

Computer Science Agile Methodologies Software Design Software Development			6. Software Engineering 7. Agile Methodologies 8. React.js 9. Redux.js 10. Angular
	Candidate 2	Current Job Position	Software Engineer @ an IT company
		Top 10 Skills	1. Node.js 2. React.js 3. JavaScript 4. Front-end Development 5. Computer Science 6. Web Development 7. Scrum 8. Agile Methodologies 9. Software Development 10. Web Services
	Candidate 3	Current Job Position	Software Engineer @ an IT company
		Top 10 Skills	1. Node.js 2. JavaScript 3. Software Development 4. Software Design 5. Algorithms 6. Front-end Development 7. Algorithm Design 8. jQuery 9. Backbone.js 10. Java

In contrast to the previous two job positions, the candidates obtained in this scenario are not high-level executives, but software engineers. Based on their skillset, all of them are experienced Front-end Developers with expertise in several JavaScript technologies. It can be observed that all candidates possess Node.js and JavaScript, while skills like *React.js, Web Development, Computer Science, Front-end Development, Agile Methodologies* and *Software Development* are only present in two candidates' profiles. Additionally, since the job position requires expertise in software design, one candidate possess this exact skill, while other has *Object-oriented design* which in fact is an approach to *Software Design*. Moreover, this skill has a direct relation to *Software Design* in the dictionary with a weight of 0.48 and an even stronger relation with *Software Development* with a weight of 0.72. In regards to the requirement for experience in an Agile environment, despite one of the candidates not possessing this skill, other skills from their profiles, such as *Software Design,* indicate a relation with *Agile Methodologies* with a weight of 0.22. A visual representation of the skills of the obtained candidates is presented in Figure 5.9.

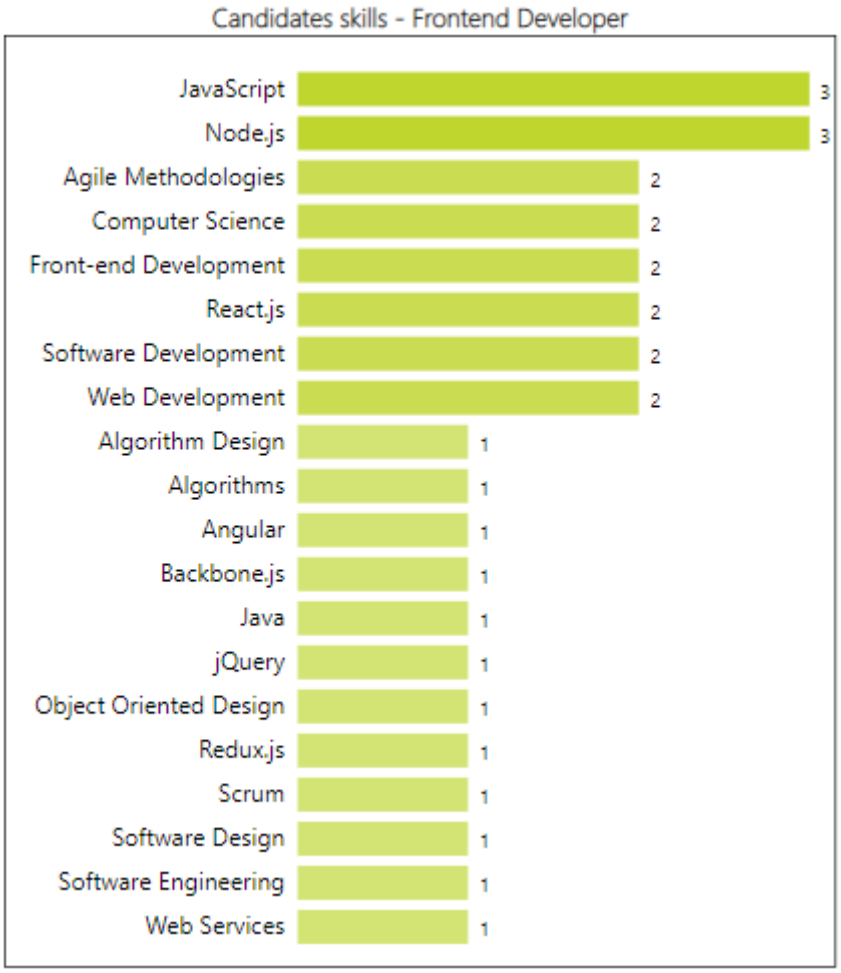


Figure 5.9 - Skill counts for the top 3 candidates for the Front-end Developer job position

These examples show fitting results based on the inputs for the job positions, however there is no reassurance if these outputs are in fact the most adequate ones from the pool of candidates. The following subsection will show the scores of the evaluation metrics obtained in the testing phase.

5.1.4. Validating a recommender system

One of the most difficult parts of recommender systems is measuring the quality of the recommendations produced. Recommender systems have been widely used in real-life applications and as such have been the topic of research for many years. However, it has been shown that there is a disparity between industry and academia in the evaluation methods of these models (Peska & Vojtas, 2020). Hence, there are two approaches in evaluating a recommender system, on-line and off-line. While academics are more focused on the traditional Machine Learning evaluation measures that do not describe the results fully, but only partially, business users find the results from the on-line experiments on live systems much more valuable.

5.1.4.1. Off-line evaluation

Off-line evaluation can be divided into two categories, implicit and explicit feedback. Implicit feedback helps estimating the results through interactions with the product, clicks, views, purchases, searches etc. Explicit feedback on the other hand, is often collected through a star rating system, which is not

always provided. This evaluation approach encompasses traditional measures from Machine Learning and Information Retrieval in order to estimate the performance of the recommendations. These measures include RMSE, Normalized Discounted Cumulative Gain (NDCG@k), Recall@k, F1 Score and Precision@k (Moosend Engineering & Data Science, 2019) and are generated based on the predictions done on the test dataset.

5.1.4.2. On-line evaluation

Unlike the off-line evaluation, online evaluation incorporates the context along with the actual needs of the user and their behaviour and interaction with the recommender system in a live environment. These experiments are often conducted through A/B Testing, which can be very time-consuming. A/B Testing is a research methodology that evaluates the user experience with the product by comparing two versions of a single variable, A and B. Depending on the context of the recommender system, various different metrics can be obtained to assess the performance, such as Click-through rate (CTR), user engagement, click rank, click-skip etc. The choice for the online measures is highly dependent on the business context in which the recommender system is used.

5.1.5. Model evaluation

This work will only include an off-line evaluation of the model's performance due to a lack of resources for further validation. The testing of the model for the utilized training mode of StarSpace is carried out by the model picking one label at random as RHS and the remaining ones as LHS for each input in the test dataset. It then proceeds to predict the top K labels for RHS given the labels in the LHS. The predictions for several test examples chosen at random from the dataset of LinkedIn profiles are shown in Table 5.8, Table 5.9 and Table 5.10.

Table 5.8 - Prediction example 1

LHS	RHS	Predictions
__label__asp.net __label__c# __label__jquery __label__xml __label__ajax __label__visualbasic.net(vb.net) __label__sql __label__cascadingstylesheets(css) __label__html __label__javascript __label__visualbasic __label__php __label__mysql __label__visualstudio __label__json __label__webservices __label__java __label__softwaredevelopment	**__label__.net**	(--) [0.704231] __label__visualbasic.net(vb.net) (--) [0.587176] __label__visualstudio (--) [0.562166] __label__c# (--) [0.558221] __label__visualbasic **(++) [0.541923] __label__.net** (--) [0.532695] __label__java (--) [0.529999] __label__sql (--) [0.495328] __label__mysql (--) [0.466303] __label__c++

Table 5.9 - Prediction example 2

LHS	RHS	Predictions	
__label__datamining			
__label__enterprisesoftware		(--) [0.765734]	__label__distributedsystems
__label__machinelearning		(--) [0.602714]	__label__datamining
__label__entrepreneurship		(--) [0.560003]	__label__artificialintelligence
__label__distributedsystems	**__label__bigdata**	**(++) [0.529039]**	**__label__bigdata**
__label__artificialintelligence		(--) [0.475364]	__label__computerscience
__label__security		(--) [0.453208]	__label__machinelearning
__label__java		(--) [0.427805]	__label__dataanalysis
__label__javascript			

Table 5.10 - Prediction example 3

LHS	RHS	Predictions	
__label__javascript			
__label__php			
__label__jquery			
__label__java		(--) [0.697342]	__label__php
__label__adobecreativesuite		(--) [0.666098]	__label__linux
__label__amazonwebservices		**(++) [0.644585]**	**__label__mysql**
__label__git		(--) [0.628341]	__label__javascript
__label__linux		(--) [0.617939]	__label__amazonwebservices
__label__sql	**__label__mysql**	(--) [0.615915]	__label__docker
__label__cascadingstylesheets(css)		(--) [0.562549]	__label__java
__label__html		(--) [0.551631]	__label__git
__label__node.js		(--) [0.541903]	__label__node.js
__label__databases			
__label__ajax			
__label__contentmanagementsystems			
__label__docker			

Since each candidate is represented as a collection of their skills, meaning that the embedding of a candidate is the average of the skills' individual embeddings, the resulting embedding could appear in the predictions for the RHS label as it could be close to the LHS embedding. An experiment was performed by using a hyperparameter named *excludeLHS.* This feature was incorporated within StarSpace as a contribution from the community post its release, with the sole purpose to exclude the labels on the LHS in the predictions. However, its functionality is questionable since the results retrieved still had labels from the LHS considered in the predictions. More examples can be seen in Appendix 9.3.

As a matter of reference, the evaluation metrics for both models are presented in Table 5.11.

Table 5.11 - Evaluation Metrics

Dataset	Hits @ 1	Hits @ 10	Hits @ 20	Hits @ 50	Mean Ranks
LinkedIn profiles	0.26%	4.68%	11.7%	26.72%	209
Resumes	0%	1.2%	1.8%	5%	365

The way these results are interpreted is that for only 0.26% of the examples, the model managed to predict the correct label as the top 1. In 4.68% of the examples, the correct label was predicted in the top 10 results and so on. Additionally, the mean rank position of 209 represents the arithmetic average position of the predicted entities from all entities, meaning that it takes into consideration the positions of the true label for each test example. As expected, the evaluation metrics for the resumes are far worse than the previous ones based on the assumptions made in this section.

5.2. DISCUSSION

This subsection will cover the experiments conducted during the training phase of the model. It will include graphs that show how the performance of the model was impacted with the tunning of certain hyperparameters. Additionally, it will include experiments that justify the implemented methodology and argue why certain decisions were made in the process.

5.2.1. Hyperparameter impact on the embeddings

Several hyperparameters were tweaked in the training phase in order to obtain the optimal model performance. Hyperparameters such as number of epochs, early stopping and batch size will not be presented as their impact is already widely known. For this reason, the experiments presented will be concerning two hyperparameters, 1) Dimensionality and 2) Negative Sampling. Since the loss function alone cannot be considered as an indicator of the quality of the embeddings produced, the decisions for the hyperparameters optimal values were chosen based on the evaluation metrics as well.

5.2.1.1. Dimensionality

The dimensionality selection process has always been an open challenge in NLP. Although there is no scientific proof on how to choose the vector size of the embeddings, some of the approaches proposed by experts in field are either doing an ad-hoc selection or through a grid search algorithm. Another empirical approach used by researchers is to do a grid search algorithm for various dimensionalities, evaluate them on a functionality test, such as word analogy, and thus proceed to choose the most optimal value (Zi & Shen, 2018). A combination of both approaches is considered in the tuning of this particular hyperparameter.

A grid search algorithm was applied for a dimensionality size of 50, 150 and 250 and the impact on the loss function is presented in Figure 5.10.

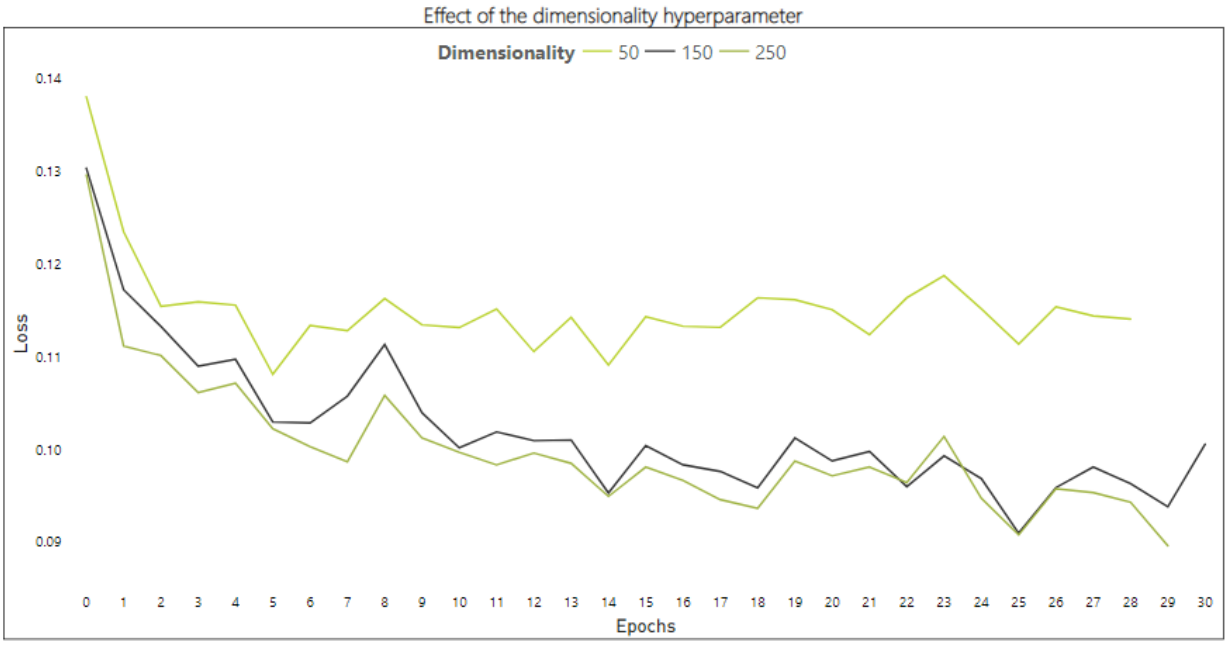


Figure 5.10 - Effect of the dimensionality hyperparameter on the loss function

The least performant dimensionality is, as expected, the smallest dimensionality of size 50. Observing the remaining two dimensionalities of 150 and 250, it can be noticed that although the latter outperforms the first in some epochs, the difference is not as significant to justify the increase of dimensions and therefore of parameters in the model. A larger vector embedding size will lead to an added unnecessary complexity to the model. Additionally, the evaluation metrics for the smaller dimensionality slightly outperformed the latter. Having this in mind, a decision was made to fixate the size of the embedding vectors to 150.

5.2.1.2. Negative Sampling

In a similar manner, the grid search algorithm was applied for the negative sampling hyperparameter for the values 50, 150 and 200. The effect of each value on the loss function is shown in Figure 5.11.

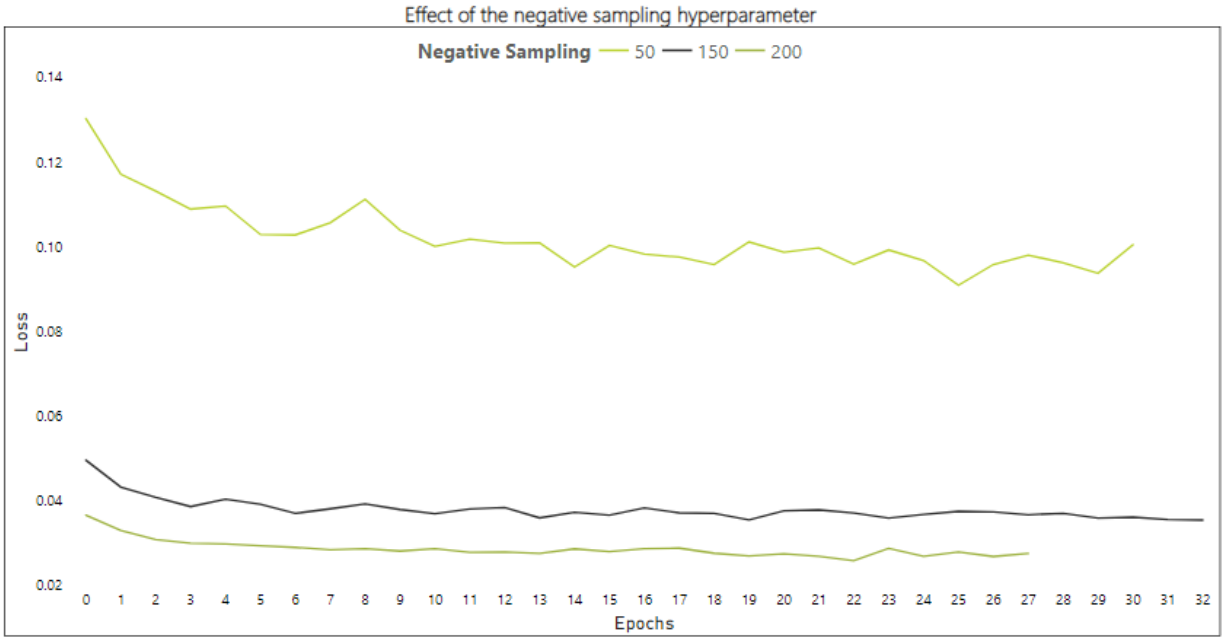


Figure 5.11 - Effect on the negative samples hyperparameter on the loss function

Although the greater values for negative samples show an improvement in the loss function, the decision for the optimal value was not as forthright as with the dimensionality hyperparameter. The experiments showed that adding more negative samples decreases the predictive power of the model, which thus led to suboptimal results. To further validate the effect that the negative sampling had on the quality of the embeddings, the nearest neighbours for the same skills and user embeddings as presented before were reproduced. The results showed a significant drop in the quality of the embeddings, thus considering 50 as the optimal number for negative samples.

6. CONCLUSIONS

This section aims to summarize the key research findings, outcomes and conclusions carried out during the course of this work. It will revise the research question raised and argue the decisions made in the proposed methodology.

The recruitment process for the right candidates relies on HR professionals' eye-scanning dozens, or even at times, hundreds of resumes to fill a certain job position. More often than not, recruiters are given a wide pool of applicants who may or may not have the required qualifications. This procedure can be time-consuming and lead to a delayed hiring process, with a risk of losing good applicants to competitors.

The main purpose of this work was to address these open issues and refine the recruitment process not only by accelerating it, but also by making it less costly in terms of resources. The proposed methodology is based on the implementation of a novelty neural embedding model developed by Facebook AI Research, known as StarSpace. StarSpace is a general-purpose model that can be utilized in various different modes, all of which cover a different type of use case in the field of ML. In the context of this application, by using either of the two independent datasets, one consisting of completely unstructured resumes in the form of documents and another consisting of semi-structured LinkedIn profiles, each candidate is represented as a bag of their skills. StarSpace then learns the embeddings of the skills and maps them in a common vectorial space where it compares them against each other. By having the skills embeddings produced, one can further represent a candidate as an average of their skills' embeddings. In the same way, a job position can be represented as an average of the skills' embeddings required. One of the advantages of the proposed methodology is that by complementing the dataset with an auxiliary dictionary, one can characterize each candidate by not only their original skillset, but also by related skills, which have a dose of similarity with the aforementioned. The idea behind the recommender system is for a given job position, a vector of required skills, to output the candidates whose embeddings are closest to the input, even if some of the candidates may or may not be represented by those specific skills, but with ones which are in close proximity in the common vectorial space. An advantage to StarSpace is that it can generalize to new candidates without the need to retrain the model. However, the drawback is that the same scenario does not apply to new skills.

Extensive data preprocessing was performed on the datasets in order to clean and normalize the text and avoid unrepresentative data going into the model. The techniques applied significantly decreased the size of the vocabulary by eliminating 98% of the noise in the dataset of LinkedIn profiles and 94% of the noise in the dataset of the resumes. Moreover, multiple experiments were carried out in the fine-tuning of the model, ranging from tweaking the hyperparameters to applying filters on the dataset's vocabulary. Like any other use case in ML, hyperparameter choices reflect the standard trade-off between bias and variance. Having said this, the conclusions carried out from this process are that a smaller number of dimensions compresses that data too much, and the model is unable to learn all the information needed. However, a greater number of dimensions also increases the number of parameters in the model and affects the performance while not showing significant improvements, thus a common ground had to be established. On the contrary, the improved performance for the negative sampling hyperparameter resulted in inferior quality of the embeddings.

The main difficulties encountered during the development of this work revolved, first and foremost, around the challenging process of parsing and correctly extracting the data from both the unstructured resumes and semi-structured LinkedIn profiles. The difficulty in parsing the resumes resulted in a very erroneous feature extraction process which led to omitting the dataset in the further development of this work. Aside from the unstructured nature of the resumes, this obstacle is also reflected on the dataset of LinkedIn profiles as it limited the possibilities for obtaining various features to further represent a candidate such as: experience, seniority level or education. Another relevant difficulty is concerning the evaluation of the model. Although the traditional statistical measures were used to assess its performance, studies have shown that they do not always represent the true usability of a model in a real-life environment. Additionally, no similar works in this field were found that would serve as a benchmark for the proposed methodology.

To conclude, StarSpace as a novelty neural embedding model, with hardly any implementations in practice, showed solid results for the problem assessed and managed to fulfil the purpose of this work. Its flexibility in entity representation can be utilized in various different contexts within the same universe. For instance, by applying the same approach, companies can be described as a bag of employees which in turn are represented as a bag of their skills, and thus proceed to compare them against other companies or even compare the given candidate with a company's general skillset, all in the same vector space.

7. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

This section will describe the limitations and flaws encountered during the implementation of this work, along with proposals for future developments that could potentially add value to the solution. The first limitation is related to the dictionary of LinkedIn skills used in the process of feature extraction. The dictionary was generated from an outdated file that contained data from the LinkedIn Topics Directory, a feature that has been discontinued since 2017. This limitation is applicable for skills which are rarer amongst candidates in the dataset. Additionally, and more importantly, aside from serving as a ground base for feature extraction, the dictionary also contains a set of related skills, along with a calculated ratio of co-occurrence, that indicates a continuity among these features in the model. Considering the constantly changing business environment, more specifically the rapid evolvement of technologies, this outdated information could lead to the model becoming obsolete with time, as it will not be fed with the relationships between certain skills and competencies that are yet to be discovered.

The second limitation of this work is related to the preprocessing of both resumes and LinkedIn profiles. Since the resumes are not unified in a single template, but are rather completely unstructured, the feature extraction process is very complex and prone to significant errors. Resume parsing of unstructured data is a very challenging task in NLP as in incorporates interpreting human language that varies significantly from one individual to another. The approach used in this work for splitting a resume in sections before proceeding with feature extraction, is far from robust in regards to unforeseen resumes that will most likely be fed with future use of the solution. Additionally, the results showed that the feature extraction process in the resumes was rather unsuccessful. Moreover, despite the lack of need to perform detection of sections in the LinkedIn profiles, as it was already given, the ability to extract relevant experience is far from simple, since people tend to be very creative with their ways of expression.

The third limitation follows up on the previous. One of the most relevant elements of a recruitment process for a given job position, aside from the competencies, is the experience of the candidate along with their seniority level, which is not incorporated in this work. The reason for the absence of this feature is mainly due to the complexity for its extraction from the datasets. Additionally, the dataset of the LinkedIn profiles only includes two previous job experiences, which can lead to wrongful information regarding the true seniority level of a candidate. The drawback of this feature can also be observed in the results for the nearest neighbors algorithm for the given job positions, where despite requesting associate level professionals, the majority of the retrieved candidates were high-level executives.

Last but not least, a very relevant limitation of this work is the online validation of the recommender system. As previously described in this work, the greatest challenge in recommender systems is measuring the quality of the recommendations. The datasets at disposal are not labeled and as such make the validation process much harder. Although the results shown through the methods of nearest neighbors, along with the traditional statistical measures for model evaluation, are solid, the nature of the problem assessed requires further validation performed through methods of real-life user interaction.

Having said all this, there are several proposals for addressing the aforementioned limitations of the system, beginning with the outdated dictionary used as a ground base in the feature extraction and feature engineering process. Regarding potentially missing skills, one approach to this limitation would be for a recruiter to manually update the existing dictionary with missing skills that are present amongst candidates. However, this approach does not apply to the issue with the outdated and missing relationships. A solution to this limitation would be to feed the model with a considerable amount of data so that it can learn the relationships between the skills independently. However, since vast amounts of data are not always available, another approach would be to create separate models where each would be concentrated in one or more similar areas of expertise. For instance, one model would be fed and trained with data from professionals in the area of marketing and sales, another for IT professionals which are purely technical, or maybe are a mixture of managerial and technical competencies, and so on. This approach would however sacrifice the model's ability to generalize at a cost of performance in specific tasks.

Continuing with the second and third limitation, concerning the structure of the data and the feature extraction process, a great contribution to this work would be a unified template of resumes, as opposed to the unstructured and semi-structured data at disposal. Having a predefined structure will lead to an improved feature extraction and a decreased margin of error in the data preprocessing phase. With this in mind, the availability of features such as previous experiences, seniority level and even education will be much more attainable. Additionally, considering that people are not equally fluent in every skill enlisted in their resume or LinkedIn profile, the proposed template could have an input field with a predefined scale where the candidate could assign the level of knowledge along with each expertise. All these features combined would enrich the model and hopefully lead to a better embedding of the candidates in the vector space, which would be a great added value to the end-users of the system.

Finally, this work is completed with a proposal of a baseline for online evaluation of the model. The evaluation would be divided into two phases where, for a given period of time the HR recruiter would have a hands-on experience with the system. The first phase would be to perform a cross-validation of the traditional approach against the recommendations produced by the system, obtain certain measures and further use these insights to fine-tune the model. The second phase consists of two approaches. One approach would be for the dataset to be initially narrowed down to a smaller subset with more adequate people given the job requirements, which will reduce the amount of work by the recruiter. The second approach is a suggestion mechanism, in which the recruiter would consider the top candidates retrieved by the system and thus proceed to interview them as plausible candidates for the role.

8. BIBLIOGRAPHY

Boiret, G., & Tapia, M. (2016, March 1). Phantombuster SAS. Paris. Retrieved from Phantom Buster: https://phantombuster.com/

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. doi:https://arxiv.org/pdf/1607.04606.pdf

Deep, M. (2020, June 30). *Surprisingly Effective Way To Name Matching In Python*. Retrieved from Towards Data Science: https://towardsdatascience.com/surprisingly-effective-way-to-name-matching-in-python-1a67328e670e

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research 12*, 2121-2159.

Gopalakrishna, S. T., & Varadharajan, V. (2019, January). Automated Tool for Resume Classification Using Sentiment Analysis. *International Journal of Artificial Intelligence and Applications (IJAIA), 10*.

Gornishka, I., Rudinac, S., & Worring, M. (2019). Interactive Search and Exploration in Online Discussion Forums Using Multimodal Embeddings. doi:https://arxiv.org/pdf/1905.02430.pdf

Gulli, A., & Pal, S. (2017). *Deep Learning with Keras.* Packt Publishing.

Guo, C., & Berkhahn, F. (2016). Entity Embeddings of Categorical Variables. doi:arxiv.org/pdf/1604.06737v1

Gutmann, U. M., & Hyvarinen, A. (2012). Noise-Contrastive Estimation of Unnormalized Statistical Models with Applications to Natural Image Statistics. *Journal of Machine Learning Research 13*, 307-361.

Ko, Y., & Seo, J. (2000). Automatic Text Categorization by Unsupervised Learning. *COLING '00: Proceedings of the 18th conference on Computational linguistics. 1*, pp. 453-459. Association for Computational Linguistics.

Ko, Y., Park, J., & Seo, J. (2002, July 18). Improving text categorization using the importance of sentences. *Information Processing and Management* .

Kulkarni, Y. (2017, May). *Text Mining 101: Mining Information From A Resume*. Retrieved from kdnuggets: https://www.kdnuggets.com/2017/05/text-mining-information-resume.html

McTear, M., Callejas, Z., & Griol, D. (2016). *The Conversational Interface.* Springer International Publishing.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. doi:arXiv preprint arXiv:1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. doi:arXiv preprint arXiv:1310.4546

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, (pp. 746-751).

Moosend Engineering & Data Science. (2019, June 10). *Medium*. Retrieved from Building a Validation Framework For Recommender Systems: A Quest: https://medium.com/moosend-engineering-data-science/building-a-validation-framework-for-recommender-systems-a-quest-ec173a24b56f

Niu, F., Recht, B., Re, C., & Wright, J. S. (2011). *HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent.* doi:arXiv preprint arXiv:1106.5730

Peska, L., & Vojtas, P. (2020). Off-line vs. On-line Evaluation of Recommender Systems in Small E-commerce. *Proceedings of the 31st ACM Conference on Hypertext and Social Media*, (pp. pp. 291-300).

Raschka, S., & Mirjalili, V. (2021). *Python Machine Learning - Second Edition.* Mumbai: Packt Publishing.

Reza, T., & Zaman, S. (2017). *Analyzing CV/Resume using Natural Language Processing and Machine Learning.* BRAC University.

Sadiq, S., Ayub, J., Narsayya, G., Ayyas, M., & Tahir, K. (2016, April). Intelligent Hiring with Resume Parser and Ranking using Natural Language Processing and Machine Learning. *International Journal of Innovative Research in Computer and Communication Egineering*.

Sanyal, S., Hazra, S., Ghosh, N., & Adhikary, S. (2017, March). Resume Parser with Natural Language Processing. *International Journal of Engineering Science*.

Schmidt, T. N. (2019, June 3). *Unsupervised Text Classification*. Retrieved from Medium: https://medium.com/@ai_medecindirect/unsupervised-text-classification-695392c6fac7

Tabrizi, S. (2017, July 24). *Scraping LinkedIn Topics and Skills Data*. Retrieved from Shawn Tabrizi: https://www.shawntabrizi.com/code/scraping-linkedin-topics-skills-data/

Van Der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research 9*, 2579-2605.

Van Essen, S. (2018). *Beerlytics: An Interactive Beer Recommender System Based on Community Reviews.* Amsterdam: Faculty of Science, University of Amsterdam.

Verma, M. (2017, January). Cluster based Ranking Index for Enhancing Recruitment Process using Text Mining and Machine Learning. *International Journal of Computer Applications, 157*.

WB Advanced Analytics. (2017, July 16). *Boosting the selection of the most similar entities in large scale datasets*. Retrieved from Medium: https://medium.com/wbaa/https-medium-com-ingwbaa-boosting-selection-of-the-most-similar-entities-in-large-scale-datasets-450b3242e618

Weston, J., Bengio, S., & Usunier, N. (2011). *Wsabie: Scaling up to large vocabulary image annotation.*

Wikipedia contributors. (2021). *Stormfront (website)*. Retrieved May 2021, from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Stormfront_(website)

Wu, L., Fisch, A., Chopra, S., Adams, K., Adams, Bordes, A., & Weston, J. (2018). StarSpace: Embed All The Things! *Proceedings of the AAAI Conference on Artificial Intelligence*, *32.* doi:arXiv:1709.03856v5

Yin, Z., & Shen, Y. (2018). On the Dimensionality of Word Embedding. doi:arXiv preprint arXiv:1812.04224

Zi, Y., & Shen, Y. (2018). On the Dimensionality of Word Embedding. doi:arXiv preprint arXiv:1812.04224

9. APPENDIX

9.1. SCRAPED LINKEDIN SKILLS, JSON EXAMPLE


```
{
   "count":302014,
   "companies":{
      "Amazon":4289,
      "Apple":1802,
      "Stanford University":1202,
      "Facebook":2103,
      "IBM":4418,
      "Microsoft":7056,
      "Accenture":1609,
      "Google":7529,
      "Carnegie Mellon University":1285,
      "Intel Corporation":2163
   },
   "topSkills":[
      "Deep Learning",
      "Pattern Recognition",
      "Natural Language Processing",
      "Scikit-Learn",
      "Reinforcement Learning",
      "Text Mining",
      "Neural Networks",
      "Recommender Systems",
      "Computer Vision"
   ],
   "skills":{
      "Python":188238,
      "Matlab":148588,
      "SQL":103841,
      "Programming":108959,
      "Algorithms":135476,
      "Data Analysis":98842,
      "C":125987,
      "Java":150995,
      "C++":163262,
      "Software Development":104490
   },
   "name":"Machine Learning"
}
```

9.2. JOB ADVERTISEMENTS

9.2.1. Data Scientist

Transformation, adaptability and innovation are part of our DNA.
We are passionate about technology and want to be part of your story.
Do you want to make your IT and Telecommunications career goals real? Do we share the same passion? You've arrived at the right place, to Smart!

What does it take to be a DATA SCIENTIST?

• Degree in Computer Engineering or similar
• Professional experience as a Data Scientist
• Experience with R, Python, Java or Scala
• Experience with non-relational database
• Good communication skills
• Flexibility and dynamism
• English

Why us?
• We make the projects we participate in real
• We love what we do, and are proud of the results of our work.
• We are simple and efficient
• We value our people
• We are a dynamic, integrity and trustworthy team

Seniority Level	**Employment Type**
Associate	Full-time
Industry	**Job Functions**
Information Technology & Services	Information Technology

9.2.2. Marketing Specialist

Company X, is looking for an important final company, in a strong moment of expansion, a / a:

Digital Marketing Specialist, with the aim of strengthening the Advertising team.

Responsibility:

- Development and optimization of SEM (Google Ads, Bing Ads) and SMM (Facebook Ads, LinkedIn Ads) campaigns from a strategic and operational point of view.

Requirements

- Excellent knowledge of Google Ads and Facebook Ads platforms;
- Excellent knowledge of Google Analytics and tracking systems (tags, pixels, definition of goals on Google Analytics);

- Excellent ability to analyze, measure and report results;
- Knowledge of JavaScript;
- Good web marketing skills (SEO, SEM, Social, Content);
- Excellent knowledge of written English and good knowledge of spoken English;
- Google Ads and Google Analytics certifications.

What we offer:

- Professional and economic classification and growth commensurate with previous experiences with direct insertion in the Company;
- Dynamic work environment, the possibility of joining a successful, fast-growing and extremely innovative group.

Seniority Level	**Employment Type**
Associate	Full-time
Industry	**Job Functions**
Staffing & Recruiting	Marketing, Sales

9.2.3. Frontend developer

About The Teams

We are hiring Frontend Engineers for the following teams:

Driver Growth

The Driver Growth team leads the acquisition and retention platform and product efforts for drivers, couriers, and other earners worldwide. We are looking for engineers in Amsterdam to grow our existing team focused on Assisted Access (Hero) and growth incentive levers (Referrals/SIP).

Web Payments

The payments team builds products that we integrate into a growing number of businesses, collaborating closely with these teams. This is an opportunity to influence and build the next generation of user-facing payments solutions from scratch, working alongside designers, data scientists, user researchers, and product managers.

What you'll do

Decision Portal

As a software engineer on the Compliance Decision Portal team, you will be driving the architecture, strategy and execution on building a Portal to enable Uber agents globally process millions of documents, reports and other earner artifacts efficiently. Your work helps Earners get on the road faster by reducing friction in the funnel.

What You'll Do

- Design, deliver and maintain systems that enable hundreds of agents to process millions of earner documents, which is a critical step in unlocking Earner access to Uber’s platform.
- Collaborate with product managers, data science and global operations teams to gather requirements.
- Partner with fellow engineers to architect, develop and scale our Decision Portal, while keeping operational issues in mind.
- Mentor and support your fellow teammates.
- Drive ongoing efficiency and reliability improvements that improve the quality of the systems.
- Write clear documentation so that other engineers can partner to contribute and deliver.

What You'll Need

- 3+ years of frontend engineering experience
- Strong CS fundamentals.
- Experience with JavaScript, Node.js or React.
- Extensive software design and development skills. Ability to learn, and adapt to new technologies and contribute in a productive environment.
- Experience working in an agile engineering environment.

We ignite opportunity by setting the world in motion. We take on big problems to help drivers, riders, delivery partners, and eaters get moving in more than 10,000 cities around the world.

We welcome people from all backgrounds who seek the opportunity to help build a future where everyone and everything can move independently. If you have the curiosity, passion, and collaborative spirit, work with us, and let’s move the world forward, together.

We are proud to be an Equal Opportunity/Affirmative Action employer. All qualified applicants will receive consideration for employment without regard to sex, gender identity, sexual orientation, race, color, religion, national origin, disability, protected Veteran status, age, or any other characteristic protected by law. We also consider qualified applicants regardless of criminal histories, consistent with legal requirements. If you have a disability or special need that requires accommodation, please let us know by completing this form.

9.3. PREDICTION EXAMPLES

For the purposes of presentation, the scores of the predictions are consisted of only 3 decimals.

LHS		RHS	Predictions
__label__projectmanagement __label__webdesign __label__businessprocessimprovement __label__customerexperience __label__editing __label__dataanalysis __label__microsoftexcel __label__customersatisfaction __label__videoproduction __label__qualityassurance __label__adobephotoshop __label__microsoftpowerpoint	__label__iosdevelopment __label__management __label__cross-functionalteamleadership __label__webdevelopment __label__adobecreativesuite __label__microsoftoffice __label__html __label__indesign __label__cascadingstylesheets(css) __label__socialmedia __label__graphicdesign	**__label__adobeillustrator**	(--) [0.647]__label__videoproduction (--) [0.598] __label__customerexperience (--) [0.566]__label__editing (--) [0.521]__label__adobephotoshop (--) [0.500]__label__indesign **(++) [0.415] __label__adobeillustrator** (--) [0.413] __label__graphicdesign (--) [0.412] __label__writing

LHS		RHS	Predictions
__label__userinterfacedesign __label__mobileapplicationdevelopment __label__userexperience __label__userexperiencedesign __label__webdesign __label__mobiledevices __label__graphicdesign __label__creativedirection __label__iosdevelopment __label__informationarchitecture	__label__androiddevelopment __label__webdevelopment __label__adobephotoshop __label__adobecreativesuite __label__microsoftoffice __label__adobeillustrator __label__html __label__indesign __label__cascadingstylesheets(css) __label__socialmedia	**__label__interactiondesign**	(--) [0.972]__label__creativedirection (--) [0.496]__label__adobephotoshop (--) [0.496]__label__adobecreativesuite **(++) [0.475]__label__interactiondesign** (--) [0.453]__label__graphicdesign (--) [0.445]__label__editing (--) [0.445]__label__editing (--) [0.430]__label__userexperiencedesign (--) [0.392724]__label__digitalstrategy

LHS		RHS	Predictions
__label__cascadingstylesheets(css) __label__drupal __label__jquery __label__webdevelopment __label__webdesign __label__businessstrategy __label__mysql __label__html __label__php	__label__adobecreativesuite __label__microsoftoffice __label__adobeillustrator __label__indesign __label__socialmedia __label__graphicdesign __label__javascript __label__contentmanagementsystems __label__git	**__label__adobephotoshop**	(--) [0.871]__label__drupal (--) [0.507]__label__php **(++) [0.452]__label__adobephotoshop** (--) [0.391]__label__webdevelopment (--) [0.390]__label__contentmanagementsystems (--) [0.389]__label__graphicdesign (--) [0.381]__label__webdesign (--) [0.379] __label__xhtml